Efficient Range and Join Query Processing in Massively Distributed Peer-to-Peer Networks

by

Qiang Wang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2008

Author's declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Abstract

Peer-to-peer (P2P) has become a modern distributed computing architecture that supports massively large-scale data management and query processing. Complex query operators such as range operator and join operator are needed by various distributed applications, including content distribution, locality-aware services, computing resource sharing, and many others.

This dissertation tackles a number of problems related to range and join query processing in P2P systems: fault-tolerant range query processing under structured P2P architecture, distributed range caching under unstructured P2P architecture, and integration of heterogeneous data under unstructured P2P architecture. To support fault-tolerant range query processing so as to provide strong performance guarantees in the presence of network churn, effective replication schemes are developed at either the overlay network level or the query processing level. To facilitate range query processing, a prefetch-based caching approach is proposed to eliminate the performance bottlenecks incurred by those data items that are not well cached in the network. Finally, a purely decentralized partition-based join query operator is devised to realize bandwidth-efficient join query processing under unstructured P2P architecture.

Theoretical analysis and experimental simulations demonstrate the effectiveness of the proposed approaches.

Acknowledgements

I would like to thank my outstanding supervisor Dr. M. Tamer Özsu. Tamer is a great leader, an approachable but sharp supervisor who carefully guides me and works with me along the six years of my Ph.D. program.

Tamer is an active researcher at a variety of fields including distributed data management, XML query processing, streaming data processing, and information retrieval. During year 2003, when we were discussing about my Ph.D. topics, the database literature was keen on the field of massively distributed large-scale peer-to-peer (P2P) query processing. Many research papers, prototypes, and practical systems were published and the beautiful theory and ideas behind some of the works greatly impressed me. So I talked with Tamer, and he also got interested in this area and encouraged me to proceed.

My first project was targeted at XML query processing over P2P networks. It was in time because the literature just witnessed the integration of XML awareness into the routing systems (*e.g.,* by Sarvega, an Intel company). I worked hard to define the research problems in this direction and bring forward potential proposals. Tamer was working together with me on the project, questioning my motivation, correcting the design principles, and helping me out with the technical report writing. Together with Tamer's summer student, Abhay Kumar Jha from IIT Bombay, we implemented a prototype of XML routing mechanism over unstructured P2P networks and conducted the performance evaluation. Not only a programming expert of Linux scripts and C++, Abhay also helped proof-read the report in multiple times to make it a more decent work. For this project, I also appreciate Dr. Ning Zhang at University of Waterloo, for his numerous help on discussing relevant XML synopsis and XML indexing techniques, which substantially improved the quality of the work.

In parallel to the research project on P2P XML query processing, I got interested in relational data and query types such as range queries over P2P networks. Range query is such a fundamental query type that already attracted substantial attentions from the literature. Here I want to thank Prof. Ihab F. Ilyas and Prof. Keshav Srinivasan, who were sitting in my dissertation committee and working closely with me on the major work regarding this project. Ihab was a fresh faculty at University of Waterloo, but he already established himself as an outstanding researcher at database ranking and auto-tuning domains. He was helping out from the beginning of the project. We held regular meetings for problem definition and technical discussions. Then I was lucky that our concern on P2P range query processing met a novel P2P project that was led by Professor Keshav Srinivasan

from network research group at University of Waterloo and Dr. André Allavena, a research fellow from Cornell University. They designed a balanced-tree-based P2P architecture that realized important design principles including scalability, robustness, and virtualization. This architecture was originally created to support aggregate query, but by deploying range-based indexes, it can be used to support range query processing without losing desirable characteristics. I was intensively involved in this project, in charge of the implementation of the overlay network architecture, the realization of range query processing, and the improvement of the configurability of the system. All our hard work made its way to the VOILA range query processing system, which constitute an essential part of my thesis work.

In addition to structured (*i.e.,* tree-based) P2P architecture, I was also interested in exploring how range query processing could be realized over purely unstructured P2P networks. Originally, the solution looked straightforward because range queries can be multicast and the query processing can be resolved locally at each peer. However, I figured out that, when data caching is employed, poorly-replicated data items may become the bottleneck of overall processing of range queries. I talked about this issue with Dr. Khuzaima Daudjee at University of Waterloo, an expert in distributed data management. He got interested in the problem as well and we worked together and came up with a feasible approach to use prefetching techniques to improve range query processing over unstructured P2P networks. I am also grateful to Dr. Reza Akbarinia from INRIA France, who gave me critical feedbacks on the written report from a reviewer's perspective. Reza also helped me on a project of DHT-based P2P join query processing. I remembered that, Tamer, Reza and I had heated discussion of this project during weekly meetings, which made the daily hard research work more interesting.

I want to thank Dr. Lei Chen and his student Rui Li from Hong Kong University of Science and Technology, and Dr. Jie Lian from ECE department of University of Waterloo. We were working cooperatively on the semantic information retrieval project over P2P networks. I also appreciate my office-mates and personal friends Huaxin Zhang and Ye Qin for their generous time and effort with me for technical discussion, improvement of paper writing and presentation skills, and many other favors. I also thank my groupmates Yingying Tao, Patrick Kling, Xin Zhan, and Amod Gupta for all the interesting technical discussions happening at regular group meetings.

I am greatly indebted to my family, who has been consistently supportive of me to complete this Ph.D. program during these years. Whenever I experienced downtime and felt frustrated, my wife Yongjuan would encouraged me to keep positive. In addition to helping out with family chores, She is an expert of professional software development with solid mathematical background, and often hinted me efficient programming methods and helped me double check mathematical derivations. Although far away at my home town Nanjing in China, my parents talk with me weekly, ensuring that I am well and inspiring me to work and proceed steadily. I also owe a lot to my parents-in-law. We got a happy but hectic time since our first baby Zerong was born December last year. They came to Canada without hesitation, giving full-time care to the family, so that I can keep the normal working pace to finish the program on time.

Finally, I want to thank the external examiners of my dissertation, Professor Ajit Singh from

Electronic and Computer Engineering department of University of Waterloo and Professor Karl Aberer from Swiss Federal Institute of Technology in Lausanne (EPFL), for their incisive comments and suggestions to improve this dissertation.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

In recent years, peer-to-peer (P2P) infrastructure has become popular in dealing with large-scale data management and query processing in massively distributed networks. Unlike traditional client-server architectures, P2P infrastructure provides a decentralized, fault-tolerant, scalable and efficient platform to support large volumes of data, large numbers of users and various applications.

According to an online media survey conducted by BigChampagne[1], the number of peers in P2P file sharing systems has hit 9.992 million in 2006 and billions of files are shared through these systems. In addition to file sharing, many other applications are deployed over P2P infrastructure, including large-scale scientific computations (*e.g.,* Seti@home[2]), distributed groupware (*e.g.,* GrooveNet[3]) and multimedia content distribution (*e.g.,* CoolStreaming[4]).

Three P2P architectures exist: unstructured, structured and super-peer-based architecture [93]. Under unstructured P2P architecture, each peer directly communicates with its neighbors in the overlay network. Various query and data shipping mechanisms have been developed, including flooding [99], gossip [53, 56, 55, 60], random walk [41] and others.

Under structured P2P architectures, peers are organized according to specific topologies (*e.g.,* ring, torus, hyper-rectangle and others) based on a certain order. Moreover, data are placed on peers in a specific way to facilitate query processing. For example, under DHT architecture, both peers and data are mapped to uniform hash identifiers. Data are placed on those peers that are responsible for the hash identifiers that are closest to the hash identifiers of data. Major DHTs include Chord [87], CAN [82], Pastry [83], Tapestry [103], Kademlia [64] and others. However, since data are arranged on peers based on uniformly-random hash keys, data locality is destroyed, making complex query processing difficult [14]. In comparison, non-hashing structured P2P architecture considers data locality in data placement so that distributed data index can be built to facilitate the processing of complex queries. Multiple non-hashing structured architectures

[1] http://www.bigchampagne.com/
[2] http://setiathome.ssl.berkeley.edu/
[3] http://www.groove.net/
[4] http://www.coolstreaming.us

have been proposed [51, 52, 48, 1, 79, 25, 44, 5].

In contrast, super-peer-based P2P architecture takes the heterogeneous capacity of different peers into consideration. It employs more powerful peers as "super-peers" to manage metadata and index information. Peers communicate with the responsible super-peers for data and query routing purposes, such as Napster[5], Gnutella[6] and Kazaa[7]. Super-peer-based architecture may inherit techniques from traditional client-server distributed architecture [74].

Under these P2P architectures, an extensive family of query operators in database literature have been studied to support various functionalities, as enumerated below.

- *Point query operator* supports exact-matching file search [87, 82, 83, 103] and distributed file management [28].
- *Aggregate query operator* is used to evaluate global states in P2P networks. For example, an approach has been proposed to measure the number of file replicas in P2P information retrieval [100]. Similarly, in P2P-based sensor networks, global statistics about sensor readings are obtained through in-network aggregate query processing [70].
- *Range query operator* is essential in retrieving the data that satisfy specified range constraints. Range-query functionality has been widely studied [51, 52, 48, 1, 79, 25, 44, 5], supporting P2P resource discovery [7] and decentralized spatio-temporal systems [68].
- *Join query operator* is important in large-scale P2P networks, where data normally originate from multiple sources, such that the data in the system may follow heterogeneous schema [42, 97]. Join queries can be used to declare these data integration tasks. In the literature, P2P join query processing has been studied in [37, 47, 77].
- *Ranking query operator* is devised for P2P networks to rank query results based on user preferences [10, 71, 65], which improves user search experience especially when the data volume (thus potentially the volume of query results) is high.
- *Similarity query operator* is essential to modern information retrieval in retrieving the data that are "correlated" to queries. Multiple research systems exist to support similarity query processing in P2P networks [52, 88, 22, 59].
- *XPath query operators* become important since XML has been widely used in data representation and exchange. XPath query operators (*e.g.,* parental axis and ancestor axis)[8] are different from relational query operators due to their manipulation of hierarchical structures (*i.e.,* XML paths). Multiple research systems have been developed in the literature [57, 38, 35, 95, 96].

[5]http://www.napster.com
[6]http://www.gnutella.com
[7]http://www.kazaa.com
[8]http://www.w3.org/TR/xpath

Among all query operators, range query operator and join query operator are important and interesting. First, they are fundamental components of relational algebra [80], acting as building blocks of other query operators (*e.g.,* rank join [49]). Second, range and join query functionalities are essential to many distributed applications, as exemplified below.

- In on-demand P2P video sharing systems (*e.g.,* Joost[9]), video clip data within a certain time frame are buffered and shared by some peers, which can be modeled as range data. Peers can pose a range query over any time frame to search the video clips that they intend to play, and the video data satisfying the range constraints are returned to the query issuer. Similarly, in a P2P location-based service system, users may request hotel information within a geographical area around a conference site, which can be modeled as a range query over the corresponding location. Range query processing may also be applied over single values as an advanced functionality of existing systems. For instance, in a music file sharing system, song files can be identified through title and release year (*e.g.,* "*Pink Floyd*", "1982"); a range query "search all Pink Floyd songs between year 1980 and 1990" will enhance the system with range-aware functionality.

- In a P2P content publication/subscription system [3, 6], each peer may act as a subscriber or publisher (or both). Subscribers declare their preferences through an attribute *pref* and publishers mark up their data with semantic categorizations, denoted by an attribute *cat*. Once the semantic categorization of certain data hosted on a publisher peer matches the preference of a subscriber peer, the latter will maintain the physical address of the former for subsequent data fetching purpose. Due to network churn, publishers may leave the network arbitrarily. Moreover, the preferences of subscribers or the categorizations of data on publishers are subject to change. All these behaviors may invalidate the current subscriptions. Thus peers should periodically update their subscriptions to improve user experiences and system performance. The exploration of up-to-date "publishers" requires an equi-join operation "$S \bowtie_{S.pref=P.cat} P$", where S is a distributed relation capturing the subscriber information while P covers all publisher information. Similarly, in mobile computing systems, on-the-road information such as Wi-Fi availability and traffic status may be collected by (different) passing vehicles [8]. An online join query "$W \bowtie_{W.location=T.location} T$" may be defined over the distributed Wi-Fi availability relation (denoted by W) and the distributed traffic status relation (denoted by T) with respect to region locations (denoted by an attribute *location*). This query may support location-aware functionalities such as "find those regions with light traffic and stable Wi-Fi availability".

In relational algebra, range query is defined as "selection" with range constraints, which can be imposed over various data types (*e.g.,* numeric and date values). Depending on whether join attribute values are required to be equal, join queries are categorized into *equi-join* and *non-equi-join* queries. A finer categorization of equi-join queries includes inner-join and outer-join queries, while

[9] http://www.joost.com

non-equi-join queries include range join, anti-join, band-join and others [80]. *In this dissertation, range queries over numeric values and equi-join queries are focused on because they are the most commonly-used query types in practice.* For conciseness, unless specified, *range query* and *join query* in the remainder of the dissertation refer to range queries over numeric values and equi-join queries respectively. In range queries, multiple constraints can be imposed through conjunction and disjunction logical operators, while in equi-join queries, multi-way join over different attributes are considered. Moreover, exact query processing (rather than approximate query processing [43]) is considered with respect to "static snapshot" semantics [10], which ignores any manipulation of query results (*e.g.,* insertion or deletion of query results) during the period of query processing.

1.2 Dissertation Scope and Contributions

This dissertation addresses join and range query processing over non-hashing-based structured and over unstructured P2P overlay networks (Figure 1.1). The importance of the range query and join query operators were discussed above. The research excludes super-peer-based P2P architecture, because well-studied query processing schemes under client-server architecture can be reused [74].

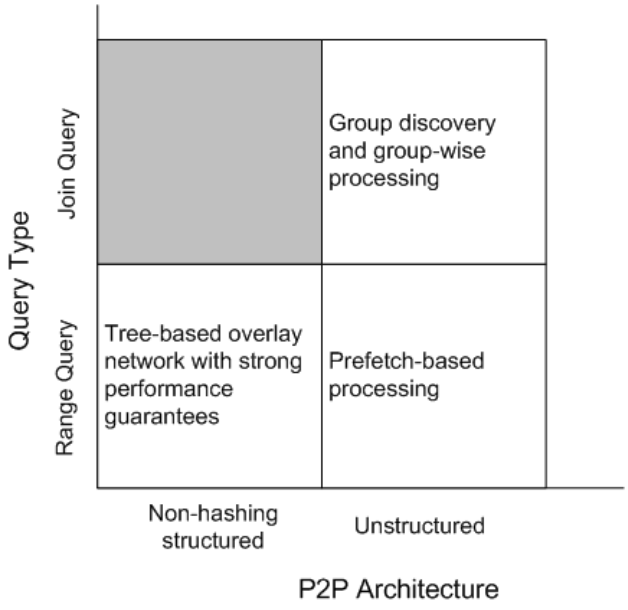


Figure 1.1: Research Scope

Note that join query processing under non-hashing structured P2P networks is not addressed, because it can be easily supported based on the approaches developed in this dissertation. For example, join attribute values can be indexed via the proposed range-aware tree-based overlay network (as shown in the lower left grid of Figure 1.1); then peers are able to execute join query processing locally. Most of the approaches developed in this dissertation can be considered above the overlay network layer, addressing functionality and quality-of-service issues in P2P query processing. Moreover, the proposed approaches are purely decentralized and scalable with respect to network

size.

In large-scale massively distributed P2P networks, query processing performance with respect to fault-tolerance, latency, and bandwidth is essential for the viability of the range and join query operators. These are the focus of this dissertation. Specifically, the following three open problems that may potentially impact query processing performance are studied:

- First, tree-structured overlay network has been widely used to support P2P range query processing [94, 91, 79, 98, 1, 25, 51, 52, 48, 8, 44]. Although existing fault-tolerance mechanisms ensure that peers are *eventually* reachable, they make no guarantees about query execution performance (*e.g.,* latency) under network churn. This is undesirable for the systems that require strong performance guarantees over range query processing, such as on-demand P2P video sharing systems (*e.g.,* Joost).

- Second, under unstructured P2P architecture, range queries can be processed at each peer by matching local data against queries. Since some data items that are requested by a range query may not be well cached within the network, the retrieval cost of these data items may be high. This may potentially cause a bottleneck for overall query execution and affect query processing performance (*e.g.,* with respect to query processing latency).

- Finally, the data in unstructured P2P networks are usually contributed by multiple sources and they may conform to heterogeneous schema. P2P join query processing can integrate these data in a decentralized way. To facilitate the query processing, effective catalog management is needed to avoid data flooding among peers during data integration. Moreover, join queries may involve large volumes of data, such that efficient query processing schemes with reduced bandwidth consumption is required.

The main contributions and an overview of the proposed approaches are presented in the following sections, each corresponding to an open problem raised in Section 1.1.

1.2.1 Range Query Processing with Strong Performance Guarantees

Tree-based overlay networks are well known for their scalability in handling range queries over P2P networks. However, existing approaches usually disregard the impact of network churn on query execution performance. For instance, when a routing hop fails due to network churn, the query is re-routed through other available links in ad-hoc fashion or it is restarted from scratch later on. This increases the number of hops, delaying query shipping and execution process.

A theoretic analysis is performed over existing tree-based overlay networks for range query processing, which reveals their incapability in providing strong query execution performance guarantees under network churn. Two novel schemes are then developed to solve the problem.

- By establishing multiple links between peers within a tree-based overlay network, a configurable number of links can be employed to ship queries based on network churn rates. Thus the proposed approach can supply strong performance guarantees for range query

processing regardless of network churn. This approach requires the change of the overlay network architecture and the routing algorithm.

- Alternatively, data and query replication schemes are developed at the query processing level to support performance guarantees over existing tree-based overlay networks. Since there are a sufficiently large number of distinct paths that exist in modern tree-based overlay networks (through auxiliary side-way links), it is feasible to issue a query in parallel from multiple peers, such that the query can be routed to destination data sources via multiple paths concurrently. This avoids retrying routing hops and guarantees the success of query shipping within a bounded number of hops. Moreover, by replicating data among multiple peers that are proximate in the overlay network, range queries can be routed via multiple distinct paths that consist of disjoint peers, effectively enhancing the fault-tolerance of the system. These replication schemes are configurable, such that they can provide query execution performance guarantees independent of varying network churn rates.

1.2.2 Data Prefetching for Distributed Range Caches

In unstructured P2P networks, flooding, gossip and random walk mechanisms are widely employed for query shipping. Due to the replication of data items (*e.g.,* via proportional replication scheme and square-root replication scheme [23]), the cost of retrieving specific data items is inversely proportional to the number of data item copies in the network. Since a range query may request multiple distinct data items, the retrieval cost of those poorly-replicated data items may dominate the query execution performance. By recognizing the correlation between poorly-replicated data items and those data that are well cached, the approach proposed in this dissertation prefetches the "correlated" poorly-replicated data items along with the requested well-cached data items. Since well-cached data items are usually popular, poorly-replicated data items can be effectively propagated by being prefetched by the queries over well-cached data items, potentially improving overall query processing performance.

The effectiveness of the approach is demonstrated theoretically by proving that, under a query distribution model with an increasing query load (refer to Section 4.3.3 for details), the approach can improve the overall performance of query execution that retrieves poorly-replicated data items by at least a factor of $\mathcal{O}(\ln m)$, where m is the query load size (*i.e.,* the number of queries over poorly-replicated data items), even when network churn and cache expiration exist. Moreover, a comparison of the proposed approach against non-prefetching schemes shows above 50% reduction of the number of messages under various query loads. The prefetch-based approach is easy to be deployed with existing replication schemes in unstructured P2P networks.

1.2.3 Partition-based Join Query Processing

Without relying on a-prior catalogs about data placement information, the approach proposed in this dissertation resorts to proactive exploration of data distribution information. Based on such information, peers self-organize into groups, each executing join queries independently

and concurrently. An effective random-sampling algorithm is developed for group membership exploration in purely decentralized fashion. Each group then applies Bloom join technique to produce join query results. The results from multiple groups are eventually aggregated to complete the query processing. Since the correctness of the query processing is guaranteed, the integration of the devised join query operator within general query processing plans is straightforward. Performance evaluation over TPC-H benchmark data[10] demonstrates that the proposed approach takes significantly less bandwidth and latency than the baseline approaches (including distributed hash join approach, symmetric semi-join approach, and Bloom-filter-based semi-join approach) that are developed in PIER system [47].

1.3 Organization

The organization of the remainder of the dissertation is as follows. Chapter 2 reviews the related work. Chapter 3 addresses range query processing with strong performance guarantees under tree-based P2P architectures. In Chapter 4, a popularity-aware prefetch-based caching approach is developed to facilitate range query processing in unstructured P2P networks. A partition-based parallel join query processing technique for unstructured P2P architecture is proposed in Chapter 5. The dissertation is concluded in Chapter 6.

[10] http://www.tpc.org/tpch

Chapter 2

Related Work

This chapter covers the works related to range and join query processing in P2P networks. Specifically, Section 2.1 focuses on the literature of P2P range query processing under tree-based overlay network architecture and unstructured P2P architecture; it also introduces the fault-tolerance and caching mechanisms that are exploited in this dissertation. Section 2.2 covers the join query processing techniques that are developed under various P2P architectures.

2.1 P2P Range Query Processing

2.1.1 Range Query Processing under DHT

The data placement in DHTs is uniform-hash-based, which may not be effective in realizing complex queries (*e.g.,* range query) [14]. Based on DHT, PIER system [47, 46] employs multicast techniques to support range query processing. Specifically, given a pair of attribute name and value, a bi-tuple key is produced, consisting of the hash id of the attribute name and the value itself. This ensures that data items with contiguous values under the same attribute name are placed on neighboring peers in DHT. During query routing, this approach first locates a "scope" that consists of all the peers with the attribute values that are requested by the range query, and then a scope-wide multicast is executed to retrieve all the data items that satisfy the range constraints.

Instead of using uniform-hashing technique, Gupta *et al* employ min-wise independent permutation technique [20, 17], which maps similar range spans to identical hash ids [43] with a certain probability. When a range query is issued, it is mapped to a query hash id and the peer responsible for this query hash id is located via DHT. Since the data hosted by the peer are expected to be proximate to range query constraint, they constitute approximate query results. In comparison to uniform-hash-based functions, this approach preserves data locality, such that it effectively supports range query processing. However, since the approach relies on approximation, it is more suitable for approximate range query processing, rather than the exact range query processing to be considered in this dissertation.

2.1.2 Range Query Processing under Non-hashing Structured Architecture

Without applying hashing-based data placement, non-hashing structured P2P architectures have also been designed for efficient range query processing. In principle, multiple distributed structures, such as tree structure, skip-list and space-filling curve, are used for range query processing.

Astrolabe [94] organizes peers into a distributed tree. Each peer at intermediate tree levels maintains a user-defined state table, and in-network aggregates are computed in multiple rounds. By aggregating the range indexes along the tree-based overlay, range query processing can be realized efficiently [15]. However, Astrolabe gains reliability through explicit leader election instead of the replication mechanisms that are considered in this dissertation.

PIER [47] is a P2P query processing system that handles a variety of SQL queries in P2P networks by using multiple separate overlay network protocols. For range query processing, PIER uses a decentralized prefix-tree-based architecture, referred to as PHT [79], which may get unbalanced when data distribution is skewed.

P-Grid [1, 29, 2] is based on a randomized binary prefix tree. However, the prefix tree becomes unbalanced under skewed data distribution so that the worst-case search cost is not logarithmically bounded. In contrast, this dissertation considers a balanced-tree-based overlay network regardless of data distribution, and the worst-case query processing cost is guaranteed to be logarithmic with respect to the network size. Moreover, this dissertation (*i.e.,* Chapter 3) focuses on improving the configurability of the tree-based overlay network to satisfy certain fault-tolerance specifications, which has not been resolved in other tree-based overlay networks in literature.

P-Tree [25] uses a B+-tree index to support P2P range query processing. However, it relies on an underlying DHT protocol (*e.g.,* Chord [87]) to deal with query routing and overlay network maintenance. Second, P-Tree can only achieve eventual consistency through running a stabilization process individually on each peer. Inaccurate range index information may misdirect query routing, affecting the query processing performance.

P-Ring system [26] employs a hierarchial routing architecture in handling P2P range queries. However, it is focused on load balancing in building range indexes, and does not address how to consistently achieve strong performance guarantees under network churn.

A series of balanced-tree-based overlay network protocols have been proposed to deal with range query processing [51, 52, 48]. BATON [51] uses a balanced binary search tree with in-level links for efficiency, fault-tolerance, and load-balancing. VBI-tree [52] enhances BATON with a limited degree of virtualization and focuses on employing multi-dimensional indexes to support more complex range query processing. Finally, BATON* [48] speeds up the query processing by increasing tree fan-out. These systems share the following common problems. First, their tree overlay structures fix the tree fan-out, such that each peer joining or leaving can cause a tree structural change, which lacks the resilience of a B-tree. Second, they employ an in-place maintenance strategy that globally updates routing and range index information immediately after any change. Under a high-churn environment, this strategy either requires a prohibitively expensive mutual exclusion mechanism to guarantee the consistency of routing table information, or tolerates considerable inconsistency.

Based on the Skip-list data structure [78], Skip-graph [8] and SkipNet [44] are proposed

as randomized balanced tree overlay network structure for query processing in P2P networks. Although they do not require explicit tree-balance maintenance procedures, either Skip-graph or SkipNet only supplies a scalable routing architecture under average case, and can degrade into an unbalanced tree in the worst case, losing query processing performance guarantees. In addition, none of them exploits locality-based clustering for performance purposes. Brushwood [101] employs Skip-graph (or SkipNet) to handle distributed data that are inherently organized as hierarchical search tree structure. The goal is quite different from the approach proposed in this dissertation, which focuses on fault-tolerant query processing with strong performance guarantees.

In addition, data items in one-dimensional space are mapped to coordinates in a multi-dimensional space through inverse space-filling curve (*e.g.,* Hilbert curve) [7]. Then peers interconnect with each other according to their spatial relationship in the multi-dimensional space so that range queries can be resolved through constrained flooding mechanisms. Conversely, multi-dimensional data items are mapped to data points in one-dimensional space through space-filling curve in [84] to support efficient query processing over multi-dimensional data.

Delay-bounded range query processing has been studied under constant-degree DHTs [58], which is different from the tree-based overlay network considered in this dissertation.

2.1.3 Fault-tolerant Range Query Processing

Fault-tolerance specifications are important in various distributed middleware [69]. Specifically, requirements on query execution performance (*e.g.,* the latency or the number of hops per query) can be declared through fault-tolerance specification. Any violation against the specification is regarded as a *performance failure* [74]. Fault-tolerance specifications will benefit P2P-based range query processing applications such as on-demand video sharing and declarative routing systems [61].

Existing overlay networks employ different fault-tolerance strategies when routing process fails due to network churn. Principally, they either retry query routing through other alive peers (*e.g.,* in BATON and PHT), or re-issue query execution from scratch later when failed links or peers (due to network churn) have been repaired by a periodic "stabilization" process (*e.g.,* in P-Tree and Astrolabe).

For instance, in BATON (similarly VBI-tree and BATON*), when the routing process towards a specific peer fails, the query is handed over to the parent of the current peer. If the parent peer is alive, it continues the routing process. In PHT, the query is re-routed to an adjacent peer by following the "thread" links at the leaf level. In P-Tree, instead of retrying the routing in place, the peer with an unusable route simply drops the query and query issuer may restart the execution later when the overlay network has been repaired by a periodic "stabilization" process. In Astrolabe, to gain fault-tolerance, each non-leaf node is monitored for failure, and, on failure detection, an alternative is elected so that the query routing can restart subsequently.

With retrying, the number of messages and latency per query may depend on network churn (failure) rates. Thus, query processing performance cannot be always guaranteed if the overlay network is not configurable with respect to network churn rates, leading to performance failures.

In this dissertation, replication mechanisms are employed to enhance the fault-tolerance of the systems. Replication mechanisms have already been employed in tree-based overlay networks such as P-Grid and Astrolabe. However, the replication mechanisms proposed in this dissertation are realized at various levels (*i.e.,* the overlay network level and the query processing level) to provide strong performance guarantees with respect to fault-tolerance specifications, which supplies extensive solutions for various overlay networks.

2.1.4 Range Query Processing under Unstructured Architecture

Under unstructured P2P architecture, peers may interconnect without taking data locality into consideration. Various communication mechanisms have been built to support routing process, such as flooding [99], gossip [56] and random walk [41] mechanisms.

Range query processing can be easily supported based on these communication mechanisms. For example, queries are propagated within the network through gossip protocol, during which peers evaluate the specified range constraints against their local data and then all peers with matching results will send them back to query issuers.

Data replication schemes have been proposed to facilitate information search process, These are classified as uniform, proportional and square-root replication schemes [23].

The uniform replication scheme (deployed in KaZaa) caches each data item at a fixed number of peers so that it preventively excludes poorly-replicated data items from the system. However, this scheme is oblivious to query distribution such that a fixed number of peers that manage the data items covered by "popular" queries may become overloaded. Moreover, the replication scheme requires strong altruistic cooperation from peers in that peers may cache data items regardless of whether they are issuing queries, which may be infeasible in uncooperative P2P environments. In contrast, with the proportional replication scheme that is employed in Gnutella, each peer only caches the results after query execution such that the caching process is triggered by query processing and only query issuers cache the results. Similarly, the square-root replication scheme [23] makes the number of cached data items proportional to the square root of the number of the corresponding queries, improving the average query processing performance with respect to constrained cache sizes. In comparison to the uniform replication scheme, the latter two schemes achieve better load-balancing by considering query distributions and do not rely on altruism of the peers. However, they disregard the caching of poorly-replicated data items, which may become the bottleneck of range query processing.

In this dissertation, a prefetch-based approach is proposed to facilitate the caching of poorly-replicated data items to improve query processing performance, which can be directly deployed with existing replication schemes. Prefetching has been used in operating systems to facilitate instruction feed to CPU by fetching all possible instructions related to a branch conditional test in advance [86]. P2P media streaming systems have also employed prefetching techniques to buffer incoming stream data for smooth playback of the stream [21]. These however, focus on the adjustment of the volume of prefetched data and are customized for specific systems.

2.2 P2P Join Query Processing

Various distributed join query processing schemes have been developed for distributed DBMS [74]. For example, a semi-join approach has been proposed under client-server settings [13]. Its variant Bloom join techniques have been developed [9, 92]. These schemes depend on the catalogs that provide data placement information of all data sources, which is hard to obtain in purely decentralized P2P networks (with network churn).

Mutant query processing approach has been proposed to handle join queries in P2P networks [75], which propagates specific query plans among peers and conducts partial query processing at each peer. The approach assumes that data placement information is explicitly specified through URLs, but does not address how to obtain such information. Similarly, a join query processing scheme has been proposed in AmbientDB system [37], where each peer keeps the data placement information of all other peers, which is only suitable for small-scale networks. Recently, a self-join query operator for P2P networks has been studied [77]. Differently, equi-join queries over multiple join attributes are considered in this dissertation.

Similar to the approach proposed in this dissertation, a partition-based join query processing technique has been proposed in [90]. The approach employs specific peers that are called "range guards" to handle join query processing. Each range guard corresponds to a sub-domain of the join attribute domain and manages a set of other peers. Those tuples whose projection over the join attribute belong to a sub-domain are replicated on the corresponding range guard. Each range guard is in charge of all the query processing that involves the data within its sub-domain. Given a range query, only those range guards that are responsible for the range constraints of a query are accessed. However, all range guards may participate in the shipping of certain queries, such that the worst-case query shipping cost is proportional to the number of range guards. Since range guards actually act as super-peers, the design principles of this approach are different from the approach proposed in this dissertation, which focuses on the purely decentralized unstructured P2P architecture. Note that, in the proposed group construction process, a representative will be chosen to conduct Bloom filtering process. This representative takes a role similar to that of a range guard. However, a representative does not replicate tuples from the peers within the group. Moreover, the process of group discovery and representative selection is carefully covered in this dissertation, while it is unclear how range guards are chosen in P2P networks.

Join query processing over DHT is studied in PIER system [47, 46]. Based on the distributed hashing functionality of DHT, multiple hash-join-based approaches, including hash-join approach, symmetric semi-join approach and Bloom-join approach, have been proposed. Since these approaches will be considered as baseline approaches in performance evaluation, their design principles are reviewed below.

- *Hash-Join Approach* A join query Q is initially populated on all peers. Each peer p scans its local tuples. If a local tuple t contains join attributes (*e.g.,* Att_1, Att_2, ..., and Att_n), p will ship t via DHT to a (remote) peer p' that is responsible for the hash id $h(Concat(Att_1, a_1, Att_2, a_2, ..., Att_n, a_n))$, where a_i is the value of t over attribute Att_i

($i \in [1, n]$), *Concat* is a function that concatenates all the arguments, and h is the DHT hash function. The ordering of join attributes for concatenation is agreed upon among all peers in advance. Peer p' then builds a hash table for each partner table over the join attributes. Thus hash join operation is conducted over all p' peers independently and concurrently.

- *Symmetric Semi-join Approach* This approach manages to reduce bandwidth consumption by applying semi-join approach. Each peer p initially conducts a projection operation of their tuples over join attributes. Then the projected values are re-hashed via the hash-join approach described above. Then only those tuples that correspond to semi-join results are retrieved to produce intermediate and final query results. This approach is close to the approach developed in this dissertation. Since semi-join requires extra communication phases to complete the query processing, the approach may potentially incur higher query processing latency.

- *Bloom-Join Approach* This approach reduces bandwidth consumption by applying Bloom filter technique [16]. Each peer p initially sends a Bloom filter bf over all its tuples (involving join attribute values) and re-hashes bf to a (remote) peer, where all Bloom filters are aggregated for each involved relational table. The aggregated Bloom filter is multicast among the peers that host the tuples of the partner table, such that those tuples that never produce query results are pruned by Bloom filtering locally at each peer. Each peer then re-hashes the remaining tuples through the hash-join approach that is addressed above. The final results are directly returned to query issuer. The Bloom filtering may help eliminate unnecessary data shipping. However, the propagation of the Bloom filters itself may increase the bandwidth consumption. Thus the overall bandwidth cost may not necessarily be saved in practice.

Chapter 3

Fault-tolerant Range Query Processing under Non-hashing Structured Architecture

P2P overlay networks are an important infrastructure to support large-scale decentralized information retrieval and query processing. Recently, tree-based overlay networks such as BATON [51], PHT [79], P-Tree [25] and others have been developed to support range query processing in large-scale P2P networks.

P2P overlay networks manage partial routing information on each peer and employ a multi-hop routing mechanism for peer communication. Since peers may join and leave arbitrarily (commonly referred to as *network churn*), fault-tolerance mechanisms are essential to ensure continuous functioning of the system.

Tree-based overlay networks either implement customized fault-tolerance mechanisms to handle network churn (*e.g.,* BATON and P-Tree), or exploit the fault-tolerance mechanism provided by underlying routing protocols (*e.g.,* PHT). For instance, in BATON, each peer maintains $\mathcal{O}(logN)$ "sideway link" (*i.e.,* long-range links at the same tree level) to keep peers well connected, where N is network size. In contrast, PHT is built physically over a DHT substrate and exploits the DHT's fault-tolerance mechanism, which also relies on multiple paths.

Although existing fault-tolerance mechanisms ensure that peers are *eventually* reachable, they make no guarantees about query execution performance under network churn. For example, in BATON, when a peer leaves ("graceful departure" is considered in this disseration), its data are taken over by its parent peer, meanwhile a recovery process is initiated to find a replacement peer among leaf-level nodes of the tree-based overlay network. During the recovery process, the sideway link referring to the departing peer is invalid, so that query processing requesting the data either waits and retries the same link after the completion of re-configuration, or detours to visit the data via peers at parent level. Since the detour may also fail, the routing process may become longer. Similarly, in DHT-based tree overlay networks (*e.g.,* PHT), when peers join and leave, re-configuration of the DHT is enforced, with a latency overhead at least logarithmic to network

size. This leaves data unreachable although they exist in the network. In contrast, P-Tree drops the query when the routing process is impacted by network churn, and restarts query execution from scratch, which delays query processing even further.

The more frequently peers join and leave, the more the routing overhead that is incurred, thus the query processing performance (*e.g.,* the number of hops or the query processing latency per query) depends on the failure probability per hop (or interchangeably the *network churn rate*). For example, under a general tree-based overlay network with uniform network churn rate, the number of hops per query is expected to be $(\frac{p \cdot (logN+1)}{(1-p)^2} + logN)$[1] (see Section 3.1 for details), where p is the network churn rate, and N is network size. When the network is static (*i.e.,* $p = 0$), overlay networks guarantee query routing hop lengths (*e.g.,* $logN$ hops per query), but they can make no guarantees under network churn when $p > 0$. Moreover, it is very difficult to configure the overlay network to adapt to different network churn rates. For instance, BATON organizes all peers in a binary-tree topology without the flexibility to configure either overlay network structure or routing algorithm. It is also unclear how to support configurability in PHT since underlying DHT is often provided as a substrate transparent to upper-level applications.

The lack of strong performance guarantees in such architectures leads to potential *performance failure* with respect to basic fault-tolerance specifications, such as *"given a range query, return complete results within specific latency (or a specified number of hops), regardless of network churn"*. This problem becomes critical when quality of service is required for query processing.

For instance, in on-demand P2P video sharing systems (*e.g.,* Joost), a tree-overlay-based distributed tracker can be built to manage range information (*e.g.,* time-frame) of buffered video clips on multiple peers. To allow real-time streaming purpose, specific video clips are expected to be located (and shared) within certain latency (or correspondingly a specific number of hops) via range queries. This poses a strong performance requirement on query execution. A similar problem is highlighted recently in declarative routing frameworks [61], which optimize P2P in-network query processing so that quality of services, such as the number of hops per query, is satisfied.

In this dissertation, two approaches are proposed to satisfy such fault-tolerance specifications. A highly configurable B-tree-based overlay network (VOILA[2]) is developed, which provides strong query execution performance guarantees independent of varying network churn rates. For the overlay architectures that lack such configurability, replication schemes are proposed at the query processing level to achieve the same goal.

The organization of the remainder of this chapter is as follows. Section 3.1 discusses fault-tolerance specification with strong performance requirements over P2P range query processing. Section 3.2 addresses the design of VOILA. In Section 3.3, replication schemes at the query processing level are proposed to satisfy fault-tolerance specifications for non-configurable tree-based overlay networks. Section 3.4 presents performance evaluation results and this chapter is concluded in Section 3.5.

[1]The base of the logarithmic function depends on specific tree structures; for example, the base equals to 2 with respect to BATON, which employs binary-tree structure.

[2]VOILA: Virtual Overlay Index tree with Locality Awareness.

3.1 Fault-tolerance Specification

Tree structures are widely used to handle range query processing efficiently. Tree-based overlay networks share the following characteristics to support efficient and fault-tolerant range query processing:

- *Scalable query processing* Routing process between peers is expected to take no more than $logN$ hops, proportional to the height of the tree-based overlay network.
- *Fault-tolerance* Fault-tolerance mechanisms, such as multiple links between nodes, are realized to handle network churn.

For clarity, the range query processing algorithm is presented in Algorithm 1. Note that the interpretation of concept of *closeness* depends on specific routing infrastructures. For example, in trie-based PHT [79], *closeness* is measured by the number of matching bits, while in BATON [51], *closeness* is defined with respect to "range distance", which can be measured as the one-dimensional Euclidean distance between the end points of given ranges.

Algorithm 1 *range_query_processing_general*($\mathcal{Q}$)

```
1: if hosted data satisfy Q then
2:    return data to the query issuer;
3: end if
4: for each routing entry e do
5:    //greedy routing
6:    if e's range index is closer to or intersecting with Q then
7:       ship Q to the peer with the IP address contained in e;
8:    end if
9: end for
```

P2P overlay networks are usually built over wide-area network environments, such that *peer failure* and *link failure* are common. Peer failure indicates that the specific peer could not provide data or routing services, which may be incurred by peer departure, or incurred by hardware and software failures of peers. Link failure indicates that an overlay network link does not route messages effectively (*e.g.,* in query shipping); it may potentially be due to overlay network re-configuration after network churn, or by physical network failure or congestion[3].

Existing systems usually disregard the impact of network failure on query execution performance, and employ a separate recovery process to repair overlay networks so that query processing "eventually completes". This may be undesirable in the scenarios that require strong query execution performance guarantees (*e.g.,* on-demand P2P video sharing). Specifically, in such scenarios, a general description of such a requirement may be declared through the following *fault-tolerance specification*, which is usually defined as the "correct" behavior of the system under specific environment settings in distributed system literature [69].

[3]Network congestion incurs time-out of communication, which leads to the malfunctioning of the overlay network link.

- *Given a range query, return complete results within T hops regardless of network churn.*

Here "complete results" include all the query results available in the network during the query processing period. For simplicity, "static snapshot" semantics [10] regarding query results is assumed. Specifically, the semantics makes the following assumptions about data insertion and deletion due to network churn during the query processing: (1) when peers join P2P network, they do not insert new query results into the system regarding the queries that are being executed; and (2) no deletion of query results occurs due to peer departure or failure. In the latter case, when peers depart or fail "gracefully", their data are taken over immediately by other peers, while when peers fail suddenly, the data hosted by them are supposed to be still retrievable from other peers.

Since in P2P systems, query and data shipping usually take multiple hops to complete, the number of hops per query is widely used as a performance metric (*e.g.,* in BATON, PHT, P-Tree and others). Specifically, a *hop* discussed in this work is equivalent to the communication between two peers through an overlay network link within a time-out interval (*e.g.,* equal to the maximum point-to-point latency in the network).

Fault-tolerance specifications may also be declared with respect to query processing latency, such as "*given a range query, return complete results within latency S regardless of network churn*". The following sampling-based approach is able to convert query processing latency to a number of hops. Each peer draws a sufficiently large number of random peers via existing sampling mechanisms [99, 41, 53] and collects the knowledge of the point-to-point latencies from them. Based on Central Limit Theorem, actual average latency per hop, denoted by avg, is approximated over a large number of independent and identically distributed samples. Suppose that the approximate average is sufficiently accurate. The expected number of hops T that corresponds to latency S can be computed by $T = \frac{S-\epsilon}{avg}$, with the error probability bounded by $e^{-\frac{2\times\epsilon^2}{\Sigma_{i=1}^{T}(Max-Min)^2}}$ based on Hoeffding inequality [45], where Min and Max are respectively the minimum and maximum latency per hop in the network. In this way, the fault-tolerance specification in terms of query latency (S) can be converted to a specification regarding the number of hops. Without loss of generality, the fault-tolerance specifications regarding the number of hops is focused on in this dissertation.

Due to the employment of tree-based routing structure, a very small T (*e.g.,* $T < logN$) may not lead to a valid specification. Conversely, when network churn rate is too high, the specification may never be satisfied. Thus this work focuses on specifications with moderate values of T under relatively low network churn rates. Specifically, $T = \mathcal{O}(logN)$ with a small constant factor is considered.

An analysis shows that existing tree-based overlay networks cannot satisfy the fault-tolerance specification because *the number of hops per query depends on network churn rates*. Consider the general retry mechanisms that are followed by many tree-based overlay networks. When routing process fails due to network churn, the query is re-routed to another peer in the network in a best-effort manner, each step taking one extra hop. Suppose that, without network churn, the aggregated number of hops to reach an arbitrary data source is equal to h ($0 < h \leq logN$). When each routing hop fails independently, it will take more hops (including retry) to complete

the query processing. Analytically, the overall number ($E_{overall}$) of hops increases to at least $E_{overall} = E_{retry} + h = \frac{p \cdot (h+1)}{(1-p)^2} + h$, where E_{retry} denotes the number of retrying hops when network churn rate is p. This is proved in Lemma 1 and plotted in Figure 3.1, where network churn rate p varies between 10% and 50% and $h = 12$. The metric may potentially be much larger when failed queries are dropped and restarted later (*e.g.,* as in P-Tree). Since existing overlay networks lack the capability to configure query execution performance based on network churn rates, they cannot always satisfy the fault-tolerance specifications.

Lemma 1. *When each routing hop fails independently with probability p, it will take $E_{overall} = E_{retry} + h = \frac{p \cdot (h+1)}{(1-p)^2} + h$ hops to complete the query processing.*

Proof.

$$
\begin{aligned}
E_{retry} &\leq \Sigma_{i=0}^{\infty} i \cdot \binom{h+i}{i} \cdot p^i \cdot (1-p)^h && (3.1)\\
&= (1-p)^h \cdot \Sigma_{i=0}^{\infty} i \cdot \frac{(h+i)!}{i! \cdot h!} \cdot p^i && (3.2)\\
&= \frac{(1-p)^h}{h!} \cdot \Sigma_{k=0}^{\infty} ((\Pi_{i=k}^{k+h} i) \cdot p^k) && (3.3)\\
&= \frac{(1-p)^h}{h!} \cdot p \cdot \Sigma_{k=0}^{\infty} \frac{d^{h+1}(p^{h+k})}{dp} && (3.4)\\
&= \frac{(1-p)^h}{h!} \cdot p \cdot (\frac{d}{dp})^{h+1} (\Sigma_{k=0}^{\infty} (p^{h+k})) && (3.5)\\
&= \frac{(1-p)^h}{h!} \cdot p \cdot (\frac{d}{dp})^{h+1} \frac{p^h}{1-p} && (3.6)\\
&= \frac{(1-p)^h}{h!} \cdot p \cdot (\frac{d}{dp})^{h+1} (\frac{p^h - 1}{1-p} + \frac{1}{1-p}) && (3.7)\\
&= \frac{(1-p)^h}{h!} \cdot p \cdot (\frac{d}{dp})^{h+1} \frac{1}{1-p} && (3.8)\\
&= \frac{(1-p)^h}{h!} \cdot p \cdot (h+1)! \cdot (\frac{1}{1-p})^{h+2} && (3.9)\\
&= \frac{p \cdot (h+1)}{(1-p)^2} && (3.10)
\end{aligned}
$$

Thus, $E_{overall} = E_{retry} + h = \frac{p \cdot (h+1)}{(1-p)^2} + h$ and the lemma holds. □

Thus, when $\frac{p}{(1-p)^2}$ is significant, the actual number of hops to complete the query processing may violate the fault-tolerance specification, leading to performance failure. In this dissertation, the problem is solved by using one of the two different ways: (1) deploying a configurable overlay network that provides strong performance guarantees despite varying network churn rates; and (2) devising fault-tolerance mechanisms at the query processing level to enhance underlying overlay networks in order to support fault-tolerance specifications. These are addressed in Section 3.2 and Section 3.3 respectively.

3.2 Configurable Tree-based Overlay Network

In this section, VOILA, a B-tree-based overlay network with range index, is described, which provides strong performance guarantees by establishing duplicate links in overlay network structure

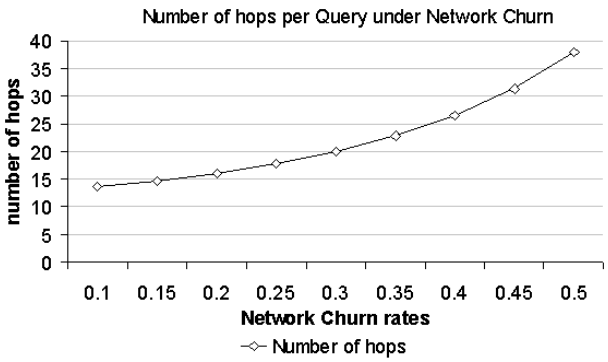


Figure 3.1: The Number of Hops per Query Under Network Churn

and employing a configurable number of them for concurrent query shipping depending on network churn rates.

The tree-based overlay network structure of VOILA is based on Leaf-Only-Tree (LOT) overlay network system that is developed at University of Waterloo [4, 5]. However, VOILA differs from LOT in that, given fault-tolerance specifications, VOILA can configure the number of links to ship range queries depending on network churn rates, rather than employing a fixed number of links as addressed in LOT. Thus the configurability of VOILA reduces the duplicate routing overhead adaptively according to fault-tolerance specifications, meanwhile preserving the strong performance guarantees for range query processing that is accomplished by LOT.

By replicating both data and routing information, VOILA effectively handles both peer failure and link failure. Thus "network failure" may be used to indicate both of them in the remainder discussion.

3.2.1 VOILA Architecture

VOILA takes a tree-based overlay architecture. At the leaf level, peers that are geographically-proximate are grouped to form a *super-leaf* (*e.g.,* F). Each super-leaf contains between l and $2l$ peers, where l is a configuration parameter. A gossip-style algorithm [36] is employed to propagate information within a super-leaf. Gossip-style algorithms are extremely tolerant to peer and link failures, take $\mathcal{O}(\ln l)$ time and use $\mathcal{O}(l \ln l)$ messages. A super-leaf can therefore be viewed as an atomic unit with a self-consistent view of the world. In the sequel, a super-leaf refers to the collection of peers belonging to the super-leaf. For example, that super-leaf F has information s means that all peers within super-leaf F have the same information s.

An example VOILA tree-based overlay is illustrated in Figure 3.2. For clarity, in the remainder, *v-node* is used to denote an upper-level node in the overlay, such as $Anc^1(F)$ and $Anc^2(F)$, which are the ancestors of peer $\mathcal{P} \in F$ at tree level 1 and 2 respectively.

Super-leaves are organized to form a *virtual tree*, where each tree node's degree is between b and $2b$ and each leaf of the tree is actually a super-leaf (the terms "leaf" and "super-leaf" will be used interchangeably). Each tree node is *emulated* by every one of its descendent leaves, which

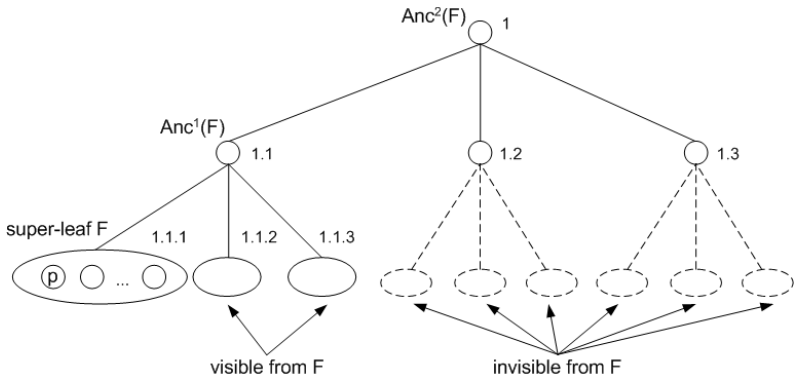


Figure 3.2: Partial Tree Representation of p

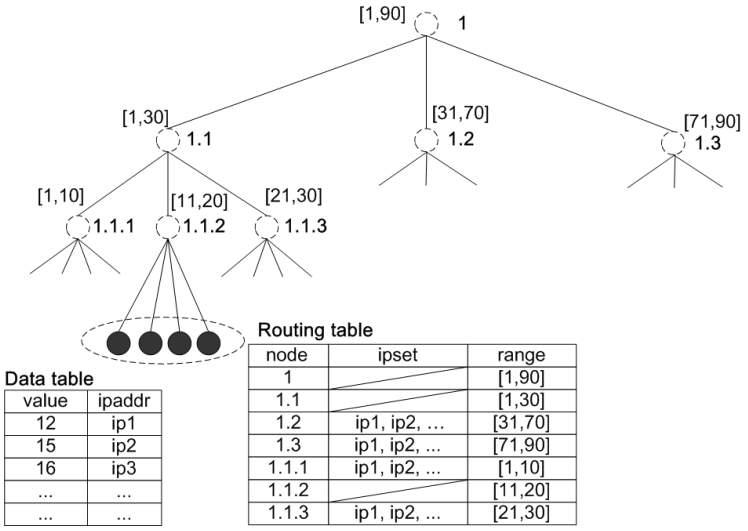
means that peers (*i.e.,* leaf nodes) act as all virtual tree nodes (referred to as v-nodes) on the path from their super-leaf to the tree root in all routing and computation tasks. Note that a peer only maintains a *partial* representation of the virtual tree, storing information about its super-leaf, its ancestors and the children of these ancestors, but not the descendants of these children. This reduces the storage requirement at each peer. As shown in Figure 3.2, the descendants of v-nodes 1.2 and 1.3 are invisible from super-leaf F.

Each peer maintains both range index and routing information for the v-nodes in its partial representation of the VOILA tree. The former state is referred to as *index state*, and the latter is referred to as *routing state*. In this dissertation, only one-dimensional range indexes are considered, although it is feasible to deploy multi-dimensional data structures (*e.g.,* minimum bounding box) to support multi-dimensional range queries. The range domain is evenly partitioned among the super-leaves and the range span of a v-node covers the span of its children.

The routing state for a v-node is defined as *a random sample of m IP addresses of peers emulating it*, where m is a configurable *duplication factor*. Every peer's routing table consists of the routing state for every v-node it knows about. Since both index and routing states are replicated on all emulating peers, by contacting any of them, peers can access the information corresponding to the v-node, which enables parallelism. In contrast to other tree-based overlay networks, where the number of "sideway" links is limited and fixed, VOILA chooses a configurable number of alternate paths with duplicate links (up to m for each hop) in query routing, which will be detailed in Section 3.2.3.

In a standard B-tree, data are placed in sorted order on leaf nodes, and higher-level tree nodes are associated with a range that covers the data values of its descendants. The proposed approach adds a layer of indirection: each peer is allowed to physically store data in more than one range. It is also responsible for indexing a contiguous range of data stored at *other* peers. It is these indices that are contiguous, and constitute the *index state* held by a VOILA peer. This design is in principle similar to the classic B+-tree alternative 2 implementation [80], which decouples the index from the real data location. This design saves the cost of clustering based on range-locality, and alleviates security or autonomy concerns caused by imperative data placement.

For instance, a peer may hold data values 1, 5 and 10, and be responsible for the range [11, 20] (*i.e.,* its index state). Then, it will have pointers to every peer (*i.e.,* by maintaining its IP address) that holds data values in the range [11, 20], and is pointed to by peers responsible for the ranges that include the values 1, 5 and 10. V-nodes at upper levels in the tree-based overlay use their index state variables to aggregate these ranges in the normal fashion. For simplicity, the range of a v-node is divided between its children with no gaps and no overlaps. Recall that peers are clustered to form super-leaves. The index ranges at the peers are therefore aggregated to form the index range of the super-leaf through gossip mechanism, which is then replicated at every peer in the super-leaf, as shown in Figure 3.3. Note that, in practice, a *data table* is used to manage the indirection information, and range index information is attached with the *routing table*.



node	ipset	range
1		[1,90]
1.1		[1,30]
1.2	ip1, ip2, ...	[31,70]
1.3	ip1, ip2, ...	[71,90]
1.1.1	ip1, ip2, ...	[1,10]
1.1.2		[11,20]
1.1.3	ip1, ip2, ...	[21,30]

value	ipaddr
12	ip1
15	ip2
16	ip3
...	...
...	...

Figure 3.3: Range Index and Data Deployment

As peers join and leave, tree structural constraints (*e.g.,* branching factor) may potentially be violated, requiring a maintenance process to preserve the consistency of the overlay network. Instead of using asynchronous "stabilization" process that only guarantees "eventual consistency" [25], a proactive synchronous process is employed that involves all peers periodically, which updates the range index information in routing tables such that peers keep consistent index information for range query processing. Peer synchronization is feasible through an approximate global clock, which can be obtained easily through a network-based time protocol such as Network Time Protocol [67], or even GPS. If no synchronization mechanisms are available , faster peers can re-synchronize when sending an "early" request by making all requests to be blocking requests.

The maintenance algorithm works in rounds with each round corresponding to one level of the underlying B-tree. All peers are assumed to simultaneously start the maintenance process at each level. During the first round (*i.e.,* round 0), all super-leaves compute the corresponding index and routing states directly. During round i ($i \geq 1$), the states of the v-nodes at level i are re-computed by all the super-leaves that emulate each v-node. More precisely, each super-leaf F learns the state information of all the visible v-nodes at level $(i-1)$. Then each F independently computes the

state of $Anc^i(F)$ that it emulates at level i. Because the computation is deterministic and inputs are the same, all super-leaves emulating the same v-node obtain the same refreshed state in parallel after the round.

To acquire the state information of v-nodes at level $(i-1)$, for each v-node v, a bounded number (*i.e.,* m) of peers belonging to super-leaf F simultaneously invoke remote procedure calls to a subset of the peers that emulate v in the routing state, fetching its index and routing states at level $i-1$. v's fan-out information is also piggybacked for classic B-tree style split-merge process [80]. When all remote procedure calls finish, peers gossip the obtained information within F. Thus all peers belonging to F acquire the complete information about v-nodes at level $(i-1)$ without further need of communication to remote peers. They then independently enforce B-tree-style split-merge mechanism to finish the structural change and the synchronization of state information at level i. The maintenance algorithm at any level i is presented in Algorithm 2 regarding an arbitrary peer $\mathcal{P}$ belonging to a super-leaf F, where C denotes branching factor constraint. Since all peers acquire consistent information about all visible v-nodes at level $(i-1)$, each peer can make globally-consistent decisions on which v-nodes merge or which v-node splits, directly implementing a decentralized consensus over the split-merging process. For example, in Figure 3.4(a), after the maintenance of the leaf level, all peers belonging to super-leaves F, G, and H will obtain consistent information about the visible v-nodes 1.2 and 1.3, including their branching factors and m sample peer addresses corresponding to each branch (*e.g.,* m samples from each of the super-leaves A through E for v-node 1.2). Suppose $b = 2$, all peers consistently learn that v-node 1.2's branching factor violates the branching factor constraint (*i.e.,* $5 > 2 * b = 4$). Thus, all of them agree to split v-node 1.2 into two v-nodes (*i.e.,* 1.2 and 1.3) at level 1, as shown in Figure 3.4(b). Denote by S_{ABC} the super-set of the m samples of super-leaves A, B, and C, and by S_{DE} the super-set of the m samples of super-leaves D and E. Then all peers agree to maintain m random samples out of S_{ABC} as the routing information to the split-out v-node 1.2, and maintain m random samples out of S_{DE} as the routing information to the split-out v-node 1.3. The splitting decision is made at all peers consistently (and concurrently) in a purely decentralized fashion, without expensive coordination among them. For presentation, the original v-node 1.3 is labeled as 1.4 after the split. However, in the implementation, each v-node has a globally-unique identifier, such that there is no need to change the labels of those v-nodes that are not involved in split-merging process.

Since all peers obtain consistent index and routing state information after the synchronization at each level, it is easy to prove that, when synchronization completes at the top level, all peers acquire a globally consistent view of the overall tree structure. The period of maintenance process is configurable, depending on application requirements.

Recall that there are three system parameters: b denotes the upper-level branching factor, l constrains the size of super-leaves (*i.e.,* leaf-level branching factor), and m is the duplication factor that decides the number of IP addresses maintained for each routing entry. When the following constraints hold: $b = \ln^{\lambda} N$, where λ is a positive constant, $l = (1+\rho)b \ln N$ and $m = (1+\rho)\ln N$, Theorem 2 holds, where p is peer (or link) failure probability. This has also been proved in [5], where a more sophisticated failure model is considered that handles peer failure and link failure

Algorithm 2 *maintenance_process*(i)

each v-node at level i (beyond F) are contacted by $m = (1+\rho)\ln N$ peers concurrently;
all peers within F propagate the obtained range index information and the branching factor x through gossip mechanism;
if $i == 1$ **then**
  $C = l$; //leaf-level
else
  $C = b$; //upper-levels
end if
$\mathcal{P}$ conducts split and merging process based on decentralized consensus for those v-nodes that violate branching factor constraints (*i.e.,* split for the v-node when $x > 2 * C$ and merge for the v-node when $x < C$);

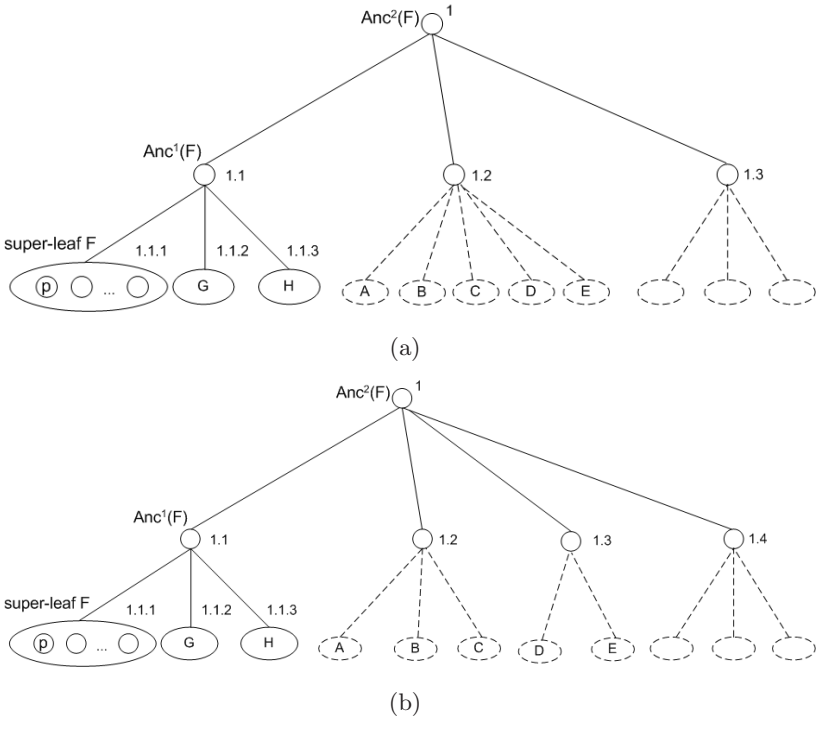


Figure 3.4: Decentralized Consensus on Split-merging Process

separately.

Theorem 2. *The overall maintenance process of VOILA fails with probability:*

$$P_{fail\_overall} = \mathcal{O}\left(\frac{1}{N^{(|\ln p|-1)+(|\ln p|)\rho}}\right)$$

Proof. Since each v-node is contacted by $m = (1+\rho)\ln N$ peers in a super-leaf F. The probability (p_{fail}) that F fails to obtain the range index and branching factor information of any visible v-node is equal to the union probability that none of the m peers that belong to F succeeds in remote procedure calls. Since all m peers issue the remote procedure calls independently, $p_{fail} = p^m = p^{(1+\rho)\ln N}$. Moreover, because there are on average $\mathcal{O}(\frac{N}{l})$ super-leaves, where N is network size and l is the branching factor constraint of super-leaves, the number of v-nodes that are visible to each super-leaf is expected to be $b \times \ln N$. Thus the union probability ($P_{fail\_overall}$) that any super-leaf fails to obtain the information of any v-node is computed as follows.

$$\begin{aligned}
P_{fail_overall} &= \mathcal{O}(\frac{bN\ln N}{l} \times p_{fail}) \\
&\leq \mathcal{O}(\frac{bN\ln N}{l} \times p^{(1+\rho)\ln N}) \\
&= \mathcal{O}(\frac{N}{1+\rho} \times p^{(1+\rho)\ln N}) \\
&\approx \mathcal{O}(e^{\ln N} \times (e^{-|\ln p|})^{(1+\rho)\ln N}) \\
&= \mathcal{O}(e^{\ln N-(1+\rho)\ln N|\ln p|}) \\
&= \mathcal{O}(\frac{1}{N^{(|\ln p|-1)+(|\ln p|)\rho}})
\end{aligned}$$

□

Thus, through the adjustment of the single parameter ρ, VOILA maintenance algorithm guarantees to run successfully with a high probability, independent of varying network churn rates.

During data updates[4], data tables of the responsible super-leaves need to be updated correspondingly. Suppose peer $\mathcal{P}$ inserts or deletes data items. $\mathcal{P}$ first issues a query to locate the super-leaf that maintains the data table covering each item. $\mathcal{P}$ then sends a notification of data table update to any peer within that super-leaf. The notification message is subsequently propagated throughout the super-leaf to update all the copies of the data table. Gossip algorithm is employed to realize the propagation due to its guaranteed efficiency and fault-tolerance.

3.2.2 VOILA-based Range Query Processing

The materialization of the generalized greedy routing mechanism presented in Algorithm 1 over VOILA is described in Algorithm 3. Conceptually, range queries are issued at the root node, and recursively forwarded down the tree to the appropriate nodes. In practice, the nodes are virtual and the request is forwarded to a randomly chosen peer emulating that v-node.

Note that query processing will fail if a peer fails while forwarding the request. On receiving a query, each peer returns a list of k peers emulating the next hop (*i.e.,* Algorithm 3 line 10), where

[4]This does not conflict with the assumption of the "static snapshot" semantics [10] in that data updates are not disallowed, only being ignored by the currently running queries.

Algorithm 3 *range_query_processing_voila*(r, i)

```
1: [Denote by i the level information, and by r the query range]
2: if i==0 then
3:    Use the data table to return the peer addresses of the entries matching range r;
4: else
5:    for each routing entry at level i intersecting with r do
6:       [Denote by V the v-node, by r_V the resulting intersection]
7:       if V is the ancestor (at height i) of this peer then
8:          call (locally) this.range_query_processing_voila(r_V, i − 1);
9:       else
10:         get a random alive peer s_V emulating V by polling k links;
11:         call (remotely) s_V.range_query_processing_voila(r_V, i − 1);
12:      end if
13:   end for
14: end if
```

$k \leq m$ is a positive integer. The enquirer can then propagate the query to these peers concurrently. Thus, the query execution plan is parallelized among multiple execution paths. As long as any one of these paths succeeds, the overall query processing will succeed. Furthermore, each partial result may be cached at the super-leaf to improve query processing performance.

When $k = m = (1 + \rho) \ln N$, the query routing is guaranteed to succeed with a high probability according to Theorem 2. Due to the employment of balanced tree structure, range query processing can complete within at most $log_b \frac{N}{l}$ hops. Thus the fault-tolerance specification (with hop constraint as T) is consistently satisfied. Alternatively, the value of k can also be configured to achieve a smaller bandwidth overhead without violating the fault-tolerance specification, as addressed in the next section.

3.2.3 Configuration of VOILA

According to Theorem 2, the probability that VOILA fails to provide consistent index and routing state information is configurable by tuning ρ, which further decides the system parameters l and m. Thus, when network churn rate p varies, ρ can be chosen so that VOILA can provide performance guarantees with a high probability (*e.g.,* $1 - P_{fail\_overall}$).

The parameter tuning can be realized either online or offline. For example, when VOILA is deployed over physical networks with different (but fixed) p, the value of ρ is pre-computed and assigned to peers when they join VOILA. On the other hand, when p changes dynamically, the value of ρ is refreshed at local peers, and propagated to all peers through a "configuration" query spanning the whole data domain, simulating a global multicast process. After obtaining the refreshed value of ρ, each peer can compute the system parameters (*i.e.,* l and m) independently but consistently, and the updated parameters will take effect during the immediate next maintenance phase, adjusting the overlay network structure adaptively.

When peers declare different fault-tolerance specifications, multiple ρ values that are configured correspondingly may be propagated to each peer. The maximum of all ρ values will be chosen by all peers, which set the system parameters to be the upper-bound values of all, satisfying all the fault-tolerance specifications in the mean time.

VOILA also configures the number (k) of links that are concurrently polled per hop according to the fault-tolerance specification (regarding T). Recall that the number of overall hops (including retry) is $E_{overall} = \frac{p \cdot (h+1)}{(1-p)^2} + h$ (see Formula 3.10 in Section 3.1), where h is the expected number of hops for query processing under static network environments. By checking k IP addresses that emulate the same v-node concurrently, an exponentially smaller failure probability for each hop can be obtained, denoted by $p' = p^k$. Correspondingly, $E_{overall} = \frac{p' \cdot (h+1)}{(1-p')^2} + h$. Thus the smallest value of k is chosen based on the following formula 3.12 to satisfy the expected number of hops per query, meanwhile achieving optimal bandwidth consumption, where $a = T - h$ for conciseness.

$$\begin{aligned} E_{overall} &= \frac{p'(h+1)}{(1-p')^2} + h \leq T \\ \Rightarrow \quad & \frac{p'(h+1)}{(1-p')^2} \leq a \Rightarrow ap'^2 - (2a+h+1)p' + a \geq 0 \end{aligned}$$

Since $p' \leq 1$ and $p' = p^k$,

$$p' \leq \frac{(2a+h+1) - \sqrt{4a(h+1) + (h+1)^2}}{2a} \tag{3.11}$$

$$\Rightarrow \quad k \geq log_p \frac{(2a+h+1) - \sqrt{4a(h+1) + (h+1)^2}}{2a} \tag{3.12}$$

Depending on specific applications, the configuration of k can also be realized either offline or online. When the fault-tolerance specification is defined with respect to all queries, the offline configuration of k acts the same as that for ρ. Instead, when different performance constraints (*i.e.,* T) are imposed over multiple queries, k can be configured at local peers during the query routing process: on receiving a query, each peer computes k according to T by using Formula 3.12, and k randomly chosen IP addresses are contacted concurrently. The latter configuration process is purely decentralized, without requiring global propagation process all over the network.

query

3.3 Replication Schemes for Non-configurable Overlay Networks

Replication schemes at the query processing level are now described for tree-based overlay networks that cannot configure query execution performance based on network churn rates, such as BATON, VBI-tree, PHT and P-Tree. The approach does not require change of routing architecture (*e.g.,* tree) or routing table organization. Instead, each peer is enhanced with a

lightweight independent random-sampling service. A minor change is also made to the routing algorithm by making routing decisions based on query shipping history records.

As discussed before, network churn incurs both peer failures and link failures. Different approaches are developed to handle different failure types in this dissertation. Specifically, when link failures are dominating, multiple paths with disjoint links are employed to improve system fault-tolerance. In contrast, when peer failure is common, distinct paths with disjoint peers can be employed. Two schemes are presented in this dissertation: a *disjoint replication link scheme* that handles link failures by shipping queries through multiple paths consisting of disjoint links (Section 3.3.1), and a *disjoint peer replication scheme* that replicates data proactively on multiple peers and employs query relaxation technique to ship queries via distinct paths consisting of disjoint peers (Section 3.3.2).

3.3.1 Disjoint Link Replication Scheme

Non-configurable tree-based overlay networks such as BATON and P-Tree do not replicate data on multiple peers. When a peer leaves, its data are taken over by a neighboring peer immediately. This re-configuration incurs nontrivial communication overhead (*e.g.,* proportional to $\mathcal{O}(logN)$ hops in BATON [51] and in Chord-based PHT [79]), leading to performance failure during re-routing. Since re-configuration of each peer is conducted separately, failure of each link (or hop) is regarded as independent of each other.

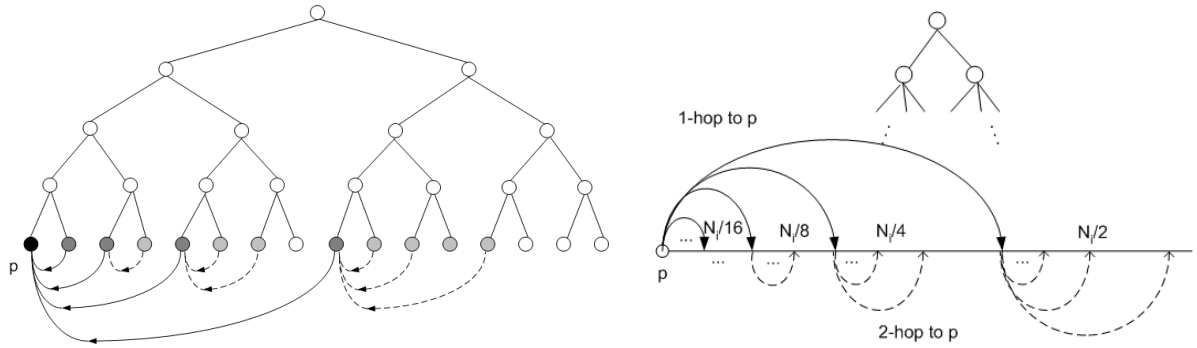
Reachability of data sources is improved by executing queries via *distinct paths* with disjoint links concurrently and by periodically retrying queries.

Existing tree-based overlay networks have two common characteristics: (1) multiple paths with disjoint links that are connected with arbitrary peers exist; and (2) the data space is usually partitioned among peers by following a certain distribution (*e.g.,* uniform distribution).

For instance, in BATON, each peer is connected with at least $\mathcal{O}(log\frac{N_i}{2})$ direct sideway links at level i, where N_i is the number of peers at this level. Each peer is also connected with more peers indirectly via sideway links recursively. An example BATON-like tree-based overlay network is illustrated in Figure 3.5(a), which contains 31 peers. Consider peer $\mathcal{P}$ at the leaf level, which is connected with peers (denoted by dark grey nodes) that are 1-hop away, and recursively connected with peers (denoted by light grey nodes) that are 2-hop away and so on. The following theorem is proved, which demonstrates that there exist sufficiently many distinct paths (with disjoint links) that are connected with any peer at the same level.

Theorem 3. *In BATON-like tree-based overlay networks, there are at least* $\mathcal{O}(\frac{log^j \frac{N_i}{2}}{j!})$ *distinct* j*-hop paths that are connected with any peer at level* i *with* N_i *peers, where* j *is upper-bounded by* $\mathcal{O}(logN_i)$*.*

Proof. As illustrated in Figure 3.5(b), at level i, there exist $logN_i$ peers, denoted by $S = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_{logN_i}\}$, that are connected directly with any peer $\mathcal{P}$. The peers belonging to S split the level into $logN_i + 1$ disjoint partitions. Within each partition, there exist peers that are 1-hop away from each peer $\mathcal{P}' \in S$ in the same manner, which in total constitute $log\frac{N_i}{2} + log\frac{N_i}{4} + log\frac{N_i}{8} + ... =$



(a) (b)

Figure 3.5: BATON Overlay and Level Splitting

$\Sigma_{x=1}^{logN_i}(log\frac{N_i}{2^x}) = \frac{log^2N_i - logN_i}{2}$ distinct 1-hop paths that are connected with peers in S, producing an equivalent number of distinct 2-hop paths that are indirectly connected with $\mathcal{P}$ via $\mathcal{P}'$. These peers split each partition into multiple sub-partitions, including one sub-partition of $\frac{N_i}{4}$ peers, two sub-partitions of $\frac{N_i}{8}$ peers, three sub-partitions of $\frac{N_i}{16}$ peers and so on.

More formally, the number of j-hop paths that are connected with any peer p with disjoint links, denoted by function S^j, is computed with the following recursive equations, where $n \geq 1$ denotes the number of peers of a partition of the peers at a specified tree level. When the overlay structure at each level does not wrap around (*e.g.,* in BATON and VBI-tree), peers located at the middle position of a level are connected with other peers from both sides. For simplicity, only the paths from the wider side are counted such that initially $n \approx \frac{N_i}{2}$. Instead, when the overlay structure at each level wraps around, initially $n = N_i$ consistently for all the peers at level i. Thus generally, $n \geq \frac{N_i}{2}$. Without loss of generality, binary tree overlay structure is presumed, such that the base of the logarithmic function equals to 2.

$$\begin{cases} S^j(n) = \Sigma_{x_1=1}^{log_2 n} S^{j-1}(\frac{n}{2^{x_1}}), & when\ j \geq 1 \\ \\ S^j(n) = log_2 n, & when\ j = 0 \end{cases}$$

Then, the following derivation holds:

$$\begin{aligned} S^j(n) &= \Sigma_{x_1=1}^{log_2 n} S^{j-1}(\frac{n}{2^{x_1}}) \\ &= \Sigma_{x_1=1}^{log_2 n}(\Sigma_{x_2=x_1+1}^{log_2 n} S^{j-2}(\frac{n}{2^{x_2}})) \\ &= \Sigma_{x_1=1}^{log_2 n}(\Sigma_{x_2=x_1+1}^{log_2 n} ... (\Sigma_{x_j=x_{j-1}+1}^{log_2 n} S^0(\frac{n}{2^{x_j}}))...) \\ &= \Sigma_{x_1=1}^{log_2 n}(\Sigma_{x_2=x_1+1}^{log_2 n} ... (\Sigma_{x_j=x_{j-1}+1}^{log_2 n}(log_2 n - x_j))...) \end{aligned}$$

Denote by $x = log_2 n - x_j$. Based on the well-known Faulhaber's formula [24].

$$\begin{aligned}\Sigma_{x_j=x_{j-1}+1}^{log_2 n}(log_2 n - x_j) &= \Sigma_{x=0}^{log_2 n - x_{j-1}-1} x \\ &= \frac{1}{2} \times \mathcal{O}((log_2 n - x_{j-1} - 1)^2)\end{aligned}$$

Asymptotically, the components with only the highest order (*i.e.,* 2) are kept.

$$\Sigma_{x_j=x_{j-1}+1}^{log_2 n}(log_2 n - x_j) \approx \frac{1}{2} \times \mathcal{O}((log_2 n - x_{j-1})^2)$$

and similarly,

$$\Sigma_{x_{j-1}=x_{j-2}+1}^{log_2 n}(log_2 n - x_{j-1})^2) \approx \frac{1}{2 \times 3} \times \mathcal{O}((log_2 n - x_{j-2})^3)$$

Iteratively, by resolving the summation and applying Faulhaber's formula recursively, an upper bound of the number of j-hop paths with disjoint links that are connected with any peer $\mathcal{P}$ at level i is:

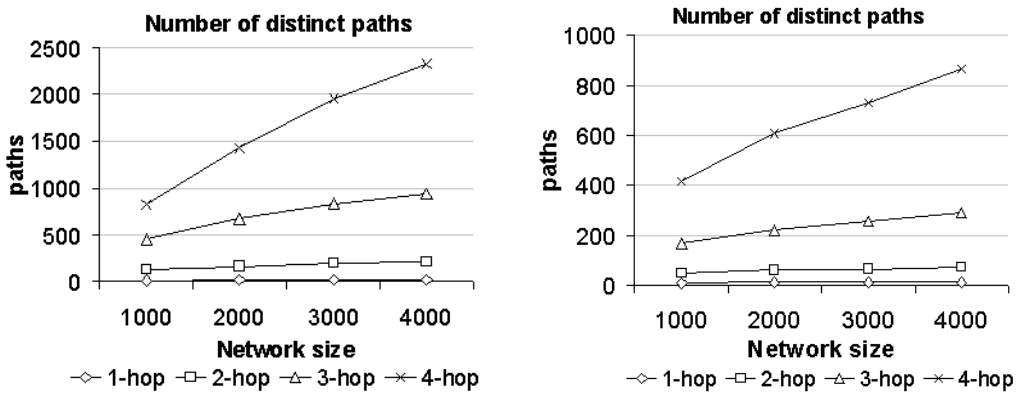
$$S^j(\frac{N_i}{2}) = \frac{1}{2 \times 3 \times ... \times j} \times \mathcal{O}(log^j \frac{N_i}{2}) = \mathcal{O}(\frac{log^j \frac{N_i}{2}}{j!})$$

□

Note that the obtained result is a loose asymptotic bound. The formula for the cases when $j = 1$, 2, 3 and 4 are detailed below for a flavor.

$$\begin{aligned}S^1(\frac{N_i}{2}) &= \mathcal{O}(log_2 \frac{N_i}{2}) \\ S^2(\frac{N_i}{2}) &= \frac{log_2^2 \frac{N_i}{2} - log_2 \frac{N_i}{2}}{2} = \mathcal{O}(\frac{log_2^2 \frac{N_i}{2}}{2!}) \\ S^3(\frac{N_i}{2}) &= \Sigma_{x=1}^{log_2 \frac{N_i}{2}-1}(x \times \frac{N_i}{2^{x+2}}) = log_2 \frac{N_i}{2} \times \Sigma_{x=1}^{log_2 \frac{N_i}{2}} x - \Sigma_{x=1}^{log_2 \frac{N_i}{2}-1}(x^2 + x) \\ &= \frac{log_2^3 \frac{N_i}{2} - 7log_2^2 \frac{N_i}{2} + 4log_2 \frac{N_i}{2}}{6} = \mathcal{O}(\frac{log^3 \frac{N_i}{2}}{3!}) \\ S^4(\frac{N_i}{2}) &= \Sigma_{x_1=1}^{log_2 \frac{N_i}{2}-1}((\Sigma_{x_2=1}^{i}) \times log_2 \frac{N_i}{2^{x_1+3}}) \\ &= log_2 \frac{N_i}{2} \times \Sigma_{x=1}^{log_2 \frac{N_i}{2}-2} \frac{x^2 + x}{2} - \Sigma_{x=1}^{log_2 \frac{N_i}{2}} \frac{(x+2) \times (x^2 + x)}{2} = \mathcal{O}(\frac{log_2^4 \frac{N_i}{2}}{4!})\end{aligned}$$

To enhance the analysis, the average number of distinct paths at the leaf level of a BATON-style tree-based overlay is measured with different network sizes (*i.e.,* 1000 to 4000 peers). As shown in Figure 3.6(a), when j grows, the average number of j-hop distinct paths that are connected with any peer increases significantly, providing sufficiently many paths for concurrent query shipping. The expected number (*i.e.,* $\mathcal{O}(\frac{log^j \frac{N_i}{2}}{j!})$) of distinct j-hop paths is also computed based on Theorem 4



(a) Number of Distinct j-hop Paths (b) Expected Number of Distinct j-hop Paths

Figure 3.6: Distinct Paths

(with the constant factor set to be 1), as shown in Figure 3.6(b), where $N = 4000$. It is not surprising that there is a linear factor difference between the actual measurement (in Figure 3.6(a)) and the expected values (in Figure 3.6(b)).

In BATON, physical peers are arranged at each level of the tree-based overlay network, which may constrain the number of distinct j-hop paths, because upper levels contain exponentially fewer paths (*i.e.,* with smaller N_i). This problem has been addressed in VBI-tree [52] by emulating upper-level (virtual) nodes with corresponding "adjacent peers" at the leaf level. Similarly, in PHT, all peers are positioned within a one-level Chord [87] ring. Consequently, by applying Theorem 3 over one level consisting of N peers, it is concluded that the number of j-hop paths that are connected with any peer is expected to be $\mathcal{O}(\frac{log^j \frac{N}{2}}{j!})$ for these virtual-tree-based systems.

It is still unclear how to locate the distinct paths that are connected with a given peer (and its data items). In existing tree-based overlay networks, the data space is usually partitioned among peers to support range indexes. For example, in BATON, newly joining peers partition the range of parent peers evenly. In PHT and P-Tree, the data domain is partitioned among peers by following uniform random distribution. The next section addresses how to locate random sample peers to issue range queries so that sufficiently many distinct paths can be employed in query shipping.

Query-Processing-Level Replication

Suppose that each link fails independently (*i.e.,* the destination peer of the hop is unreachable) with probability p. Then, the failure probability of a j-hop path is $p_{path} = 1 - (1 - p)^j$, where j is a positive integer. If each data item requested by a range query is accessed via r distinct j-hop paths, the probability that all paths fail to locate the data item is expected to be p_{path}^r, where r is referred to as *replication factor*. The above analysis on the failure probability of the j-hop path assumes that each distinct path contains j hops that are distinct from any other hops in other distinct paths. By adjusting the replication factor based on network churn rates, the proposed approach can satisfy the number of hops per query required by the fault-tolerance specification.

For example, in Figure 3.5(a), $r \leq 7$ peers denoted by light grey nodes can be chosen to reach peer $\mathcal{P}$ via r 2-hop distinct paths.

Recall that multiple paths are connected with any peer at the same level in the overlay network and the value of j is a configurable system parameter. Since the knowledge about those peers that are j-hop away from a specified peer is unknown, by randomly sampling $\mathcal{O}(\frac{rNj!}{log^j \frac{N}{2}})$ times, r peers are expected to be obtained with (distinct) j-hop paths that are connected with a specific peer $\mathcal{P}$ (and its data items) (see Theorem 4). Note that, in the proposed scheme, j-hop paths are explored during query execution, without storing path information at peers. When a range query involves more than one data items, random peers are chosen independently (and concurrently) with identical replication factors for each distinct data item.

Theorem 4. *By randomly sampling $\mathcal{O}(\frac{rNj!}{log^j \frac{N}{2}})$ times, where N denotes the number of peers at a specific level, r peers are expected to be sampled with (distinct) j-hop paths that are connected with any peer $\mathcal{P}$ at this level.*

Proof. According to Theorem 3, there exist at least $\mathcal{O}(\frac{log^j \frac{N}{2}}{j!})$ peers (denoted by S) that are j-hop paths away from peer $\mathcal{P}$ at a specific level with N peers. Thus the expected number of random samplings to obtain a peer belonging to S equals to $\mathcal{O}(\frac{Nj!}{log^j \frac{N}{2}})$. It is then straightforward that, the expected number of random samplings to choose r peers that belong to S is $\mathcal{O}(\frac{rNj!}{log^j \frac{N}{2}})$. □

To locate sufficiently many random sample peers to issue range queries, the following mechanisms are considered with respect to different data distributions.

- *Uniform Data Distribution*

  When the data space is uniformly partitioned among peers, a query-processing-level random sampling mechanism is developed by issuing queries that involve randomly-chosen range spans. Departing from the replication strategy employed in VOILA, the mechanism is realized through the range query facility provided by existing tree-based overlay networks, without changing routing algorithms of the underlying overlay networks.

  Suppose that each peer is expected to host a uniform range span $\tau = \frac{D}{N}$, where D is the data range and N represents network size. The destination peer $\mathcal{P}_{dst}$ corresponding to a random range that spans τ is therefore a random sample. To reduce random sampling cost, random sampling can be run locally as a periodic daemon process by all peers.

  Since the shipping cost per range query is $\mathcal{O}(logN)$, where N is the network size, the overall random sampling cost is proportional to the number of samples multiplied by $\mathcal{O}(logN)$.

- *Non-uniform Data Distribution*

  The data placement on peers may follow a non-uniform distribution due to various reasons (*e.g.,* load-balancing). The approach addressed below realizes an approximate scheme to draw uniformly random sample peers when the data distribution is non-uniform.

Briefly, each peer independently issues "exploratory" range queries and collects the number (referred to as "cardinality") of peers within multiple ranges. Then kernel density estimation [85] is run by each peer and a density function is computed. Peers then employ the estimated density function to synthesize range queries that can sample peers uniform randomly.

Specifically, the start point of each exploratory query is randomly chosen within the data range D, which is known in advance by all peers. The range span of each query uniformly equals to τ, which is configurable and equals to $\frac{D}{N}$ in this work, where N is the network size. Each peer issues a configurable number (n) of exploratory range queries, where the value of n affects the precision of the approximation. The cardinality of peers that are located within the range span corresponding to the query is then recorded. Based on the collected information, kernel density estimation is applied to generate an approximate density function. Specifically, the commonly-used Gaussian kernel function $K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$ is employed, and the estimated probability density function (PDF) is $PDF_k(x) = \frac{1}{Hh}\Sigma_{i=1}^{H} K(\frac{x-x_i}{h})$, where H is the number of history record samples, x_i is the value of the i_{th} sample, and h is a configurable smoothing factor.

Based on the estimated PDF, each peer chooses a random sample peer in the network by posing a random range query in the following way: a start point x is chosen with the inclusion probability proportional to the corresponding peer cardinality $PDF(x) = \frac{PDF(x)}{\int_1^N PDF(y)dy}$. After the start point is chosen, two ways are feasible to obtain a range query corresponding to a random sample: (1) based on an assumption that data are distributed uniform randomly among peers within the range span starting from x, the range span is partitioned into $\frac{\tau}{PDF(x)}$ disjoint segments and a random segment s is chosen as the range query, which is expected to correspond to a uniform sample peer in the network; and (2) without uniformity assumption, a synthesized range query $[x, x+\tau]$ is initiated and the IP addresses of $PDF(x)$ peers are expected to be returned, among which a random peer is chosen as a sample.

The proposed approach is purely decentralized and deployed directly at the query processing level. In the case that tree-based overlay networks provide aggregation facilities that compute global histograms about peer cardinality with respect to data ranges (which can be computed iteratively up the tree hierarchy), histograms can directly replace the estimated PDF in the sampling process. An approximate histogram can be acquired via the approaches proposed in Mercury [14] and Oscar [40] overlay networks, which are based on random sampling mechanism similar to the approach proposed above. Olken *et al* have also proposed a mechanism to sample tuples in B+-tree regardless of tuple distribution among disk pages [72]. The mechanism requires traversal of the tree structure from root to leaf level for each sampling, which may not be efficient for modern tree-based overlay networks because: (1) the traversal-based process needs to be realized by each peer, requiring nontrivial changes of underlying P2P architecture; and (2) the traversal from root to leaf level may potentially incur efficiency and load-balancing problems since it is oblivious to the auxiliary overlay structures (*e.g.,*

sideway links) that are supplied by modern tree-based overlay networks.

Remember that the fault-tolerance specification requirement with respect to the number of hops per query (*i.e.,* T) may tolerate to execute the query in multiple times, denoted by $c = \frac{T}{j}$. The failure probability of a specific path is p_{path} and then the failure probability after c retries decreases to p_{path}^{c}. Suppose that query processing is concurrently executed via at least r distinct j-hop paths away from a specific data source, the probability that a data item (requested by a range query) is unreachable equals to $p_{path}^{c \cdot r}$. Obviously, given c, by adjusting the replication factor r, with high probability the reachability of each data item can be guaranteed independent of varying network churn rates.

Overlay Network Changes for Range Query Processing

The underlying routing algorithm needs to be changed, which ensures that distinct paths with disjoint links are used: (1) each query $\mathcal{Q}$ is identified with a globally-unique id; each peer maintains a query-forwarding history record about the use of links (between its neighboring peers and itself) for forwarding each distinct query; and (2) on receiving query $\mathcal{Q}$, each peer selects a fresh link that has not been used to forward $\mathcal{Q}$ based on the history record obtained from neighboring peers.

Generally, range query processing consists of two phases. During the first phase, query $\mathcal{Q}$ is deployed onto $\mathcal{O}(\frac{rNj!}{log^j \frac{N}{2}})$ randomly chosen peers concurrently (*e.g.,* through multicast). All these peers then execute query $\mathcal{Q}$ via distinct paths periodically for c times, where query routing is handled by the underlying overlay networks. The range query processing algorithm enhanced with the disjoint link replication scheme is described in Algorithm 4. Let θ denote the probability with which the fault-tolerance specification is not satisfied. The value of θ is configurable depending on user preferences and is usually very small. When the other parameters (*i.e.,* p, j and T) are given, the configurable replication factor r is calculated below:

$$p_{path}^{c \cdot r} = \theta \quad \Rightarrow (1-(1-p)^j)^{c \cdot r} = \theta \quad \Rightarrow r = \frac{log_{1-(1-p)^j}\theta}{c} = \frac{j log_{1-(1-p)^j}\theta}{T}$$

Algorithm 4 *range_query_processing_disjoint_link*($\mathcal{Q}$)

1: deploy $\mathcal{Q}$ at $\mathcal{O}(\frac{rNj!}{log^j \frac{N}{2}})$ randomly chosen peers;
2: //concurrently
3: **for** each randomly chosen peer $\mathcal{P}$ **do**
4: issue $\mathcal{Q}$ periodically for c times via distinct paths (based on query-forwarding history record);
5: the underlying overlay network ships $\mathcal{Q}$ and all the data results are shipped back to the query issuer;
6: **end for**

Each query may be shipped to the peer (referred to as "destination peer") hosting query results via multiple paths. To avoid duplicate returning of query results, the destination peer records whether the results regarding each query (identified with a unique id) have been shipped to the query issuers and refrains from sending back redundant results for the same query.

3.3.2 Disjoint Peer Replication Scheme

In addition to the disjoint link replication scheme, another scheme is now proposed that employs multiple paths with disjoint peers to handle peer failures, which uses proactive data replication and depends on sideway links of tree-based overlay networks, such as BATON and P-Tree.

Sideway links constitute distinct paths with disjoint peers among "interlacing" peers in tree-based overlay networks. By replicating data items among multiple neighboring peers and employing data replication and *query relaxation* techniques, queries can be shipped concurrently via multiple distinct paths with disjoint peers. Due to data replication, this scheme directly supports *K-fault-tolerance* with respect to "peer failure" [33], where the number of replicas for each data item is configured to be $K + 1$.

Distinct paths are considered at the leaf level. This is meaningful for tree-based overlay networks with all physical peers located at the leaf level, such as VBI-tree and PHT. For others where peers are also located at intermediate levels (*e.g.,* BATON), each peer has an "adjacent peer" at the leaf level by following in-order tree traversal and these "adjacent peers" will be employed in data replication. For simplicity, data space is assumed to be uniformly partitioned among peers.

"Interlacing" Peers and Horizon

Two observations are exploited over existing tree-based overlay networks: (1) greedy routing algorithms are employed for range query processing, where queries are shipped to peers whose corresponding ranges are closer with respect to queries; and (2) due to the existence of multiple sideway links, routing paths with disjoint peers exist between"interlacing" peers.

Consider two pairs of peers $(\mathcal{P}_1, \mathcal{P}_2)$ and $(\mathcal{P}'_1, \mathcal{P}'_2)$ that are located contiguously at the leaf level. $(\mathcal{P}_1, \mathcal{P}'_1)$ and $(\mathcal{P}_2, \mathcal{P}'_2)$ are called "interlacing" peers because they interlace at the same level. As shown in Figure 3.7, between $(\mathcal{P}_1, \mathcal{P}'_1)$ and $(\mathcal{P}_2, \mathcal{P}'_2)$, there are two "interlacing" distinct paths (with disjoint peers) that consist of sideway links. Denote by x the number of peers between $\mathcal{P}_1$ and $\mathcal{P}'_1$ (respectively between $\mathcal{P}_2$ and $\mathcal{P}'_2$). A query $\mathcal{Q}$ that is initiated from peer $\mathcal{P}_1$ is first routed to the peer that is $2^{log_2 x - 1} = \frac{x}{2}$ away and then iteratively, routed to peers with decreasing distances (*e.g.,* $\frac{x}{4}$, $\frac{x}{8}$ and others) until $\mathcal{P}'_1$ is reached. Similarly, such a routing path exists between $\mathcal{P}_2$ and $\mathcal{P}'_2$ as well. Generally, the above observation holds for the tree-based overlay networks, where peers are distributed uniformly within data space and are interconnected through "long-range" links, as proved in Lemma 5.

Lemma 5. *Given two distinct pairs of peers $(\mathcal{P}_1, \mathcal{P}'_1)$ and $(\mathcal{P}_2, \mathcal{P}'_2)$, where $\mathcal{P}_1$ and $\mathcal{P}_2$ are contiguous in the tree-based overlay network at the same level and so are $\mathcal{P}'_1$ and $\mathcal{P}'_2$. Via sideway links (or long-range links), the routing path used to ship a query between $\mathcal{P}_1$ and $\mathcal{P}'_1$ consists of peers that are disjoint from those contained by the routing path between $\mathcal{P}_2$ and $\mathcal{P}'_2$.*

Proof. Suppose $\tau = \frac{D}{N}$, where D denotes the data space domain and N is network size. Due to the employment of even data partition (as in BATON) or uniform hashing (as in DHT-based P-tree), each contiguous segment with range span τ is expected to contain one peer. Suppose that $\mathcal{P}_1$ (respectively its immediate neighboring peer $\mathcal{P}_2$) intends to ship a query $\mathcal{Q}$ to a peer that is

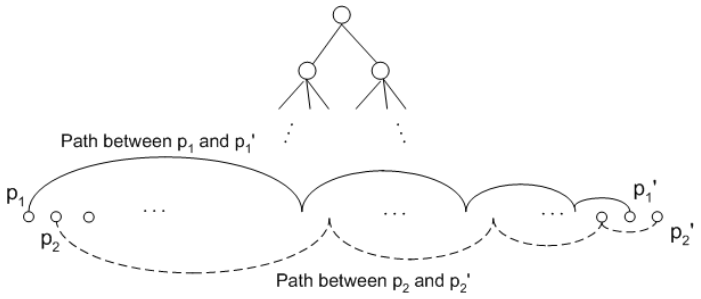


Figure 3.7: Interlacing Peers and Distinct Paths

x distance away. Starting from $\mathcal{P}_1$, $\mathcal{Q}$ is first shipped via a sideway link to a peer that is 2^{logx-1} distance away. Similarly, $\mathcal{P}_2$ uses its sideway link to ship $\mathcal{Q}$ to another peer that is 2^{logx-1} distance away. Since the range gap between $\mathcal{P}_1$ and $\mathcal{P}_2$ equals to τ, their sideway links are expected to be connected with disjoint peers. Iteratively, $\mathcal{Q}$ will be shipped via a sideway link to a peer that is 2^{logx-2} distance away from $\mathcal{P}_1$ (and $\mathcal{P}_2$ respectively). Consequently, the paths that are used to ship $\mathcal{Q}$ consist of disjoint peers. □

Remember that tree-based overlay networks (*e.g.,* BATON, VBI-tree, PHT and others) have sorted-access traversal functionality that allows to traverse peers that are contiguous in the overlay network by following a certain order. For example, in BATON and VBI-tree, the traversal functionality is enabled through "*adjacent peers*", while in PHT, such functionality is enabled through "*thread link*".

Consider peer $\mathcal{P}$ that holds data items satisfying query $\mathcal{Q}$. $\mathcal{P}$ maintains the IP addresses of a configurable number of leaf-level proximate peers, which hold contiguous range values. These peers constitute the *horizon* of $\mathcal{P}$.

Figure 3.8 shows the horizon of peer $\mathcal{P}$ that consists of 3 leaf-level peers. In BATON, it is expected to take $3 \times 2 = 6$ hops to locate the horizon via traversal, while in VBI-tree, it is expected to take just 3 hops to locate the horizon since virtual peers at intermediate levels are emulated by "adjacent" physical peers at the leaf level.

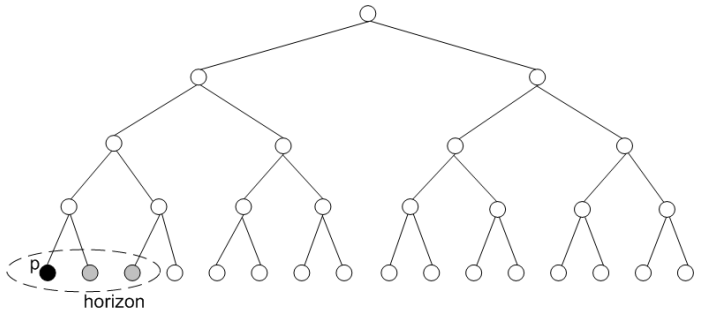


Figure 3.8: Horizon of Peer $\mathcal{P}$

Horizon membership information is refreshed periodically at each peer independently, which is

inexpensive with respect to communication cost because these peers are inherently proximate in the overlay network.

Proactive Data Replication and Query Relaxation

Each peer $\mathcal{P}$ replicates its data on the peers in the horizon. The number (*i.e.,* replication factor r) of replicas is configured based on network churn rates and fault-tolerance specification (*e.g.,* with respect to the number of hops T)[5].

Recall that network churn rate (*i.e.,* peer failure rate) is p. Since each path between $\mathcal{P}$ (*i.e.,* the query issuer) and data items consists of $\mathcal{O}(logN)$ links on average, r can be computed using the following formula, where θ denotes the probability with which the fault-tolerance specification is not satisfied and $c = \frac{T}{\mathcal{O}(log_2 N)}$ is the expected number of times that $\mathcal{P}$ can issue (or retry) query shipping:

$$p_{path}^{r \times c} = \theta \quad \Rightarrow (1 - (1-p)^{logN})^{r \times c} = \theta \quad \Rightarrow r = \frac{log_{1-(1-p)^{logN}} \theta}{c}$$

Remember that the range index information of a peer is maintained in routing tables of other peers in the overlay network. The data replicas that are hosted by the peers in $\mathcal{P}$'s horizon do not affect the range index information. Without relying on existing routing algorithm, $\mathcal{P}$ exploits query relaxation techniques to use these data replicas, as described next.

Suppose that peer $\mathcal{P}$ issues query $\mathcal{Q} = (start, end)$, where $end \geq start$ and $end - start \leq \frac{D}{N}$. When $end - start > \frac{D}{N}$, $\mathcal{Q}$ can be evenly decomposed into $\frac{(end-start) \times N}{D}$ sub-queries, which are then populated to $\frac{(end-start) \times N}{D}$ independent peers in the network. Thus the process to be addressed below can be directly applied over each sub-query respectively.

$\mathcal{Q}$ is relaxed into $\mathcal{Q}' = (start, start + r \times \frac{D}{N})$ at peer $\mathcal{P}$, where r is the replication factor to satisfy the fault-tolerance specification[6]. $\mathcal{Q}'$ is evenly divided into r sub-queries, with each sub-query populated in order to one peer in $\mathcal{P}$'s horizon. Since each sub-query of $\mathcal{Q}'$ is issued by a distinct peer, and the sub-queries are initiated by contiguous peers in $\mathcal{P}$'s horizon, one distinct path consisting of disjoint peers is expected to route each sub-query, according to Lemma 5. Thus r distinct paths will be employed to ship $\mathcal{Q}'$. Let the original query $\mathcal{Q}$ be piggybacked with the routing of each sub-query of $\mathcal{Q}'$, $\mathcal{Q}$ is then expected to arrive at r destination peers, all of which happen to host data (copies) within $(start, end)$ (*i.e.,* the results corresponding to $\mathcal{Q}$). This process ensures that query $\mathcal{Q}$ is shipped via r distinct paths with disjoint peers, as formally proved in Theorem 6.

Theorem 6. *By employing the distinct peer replication scheme, query $\mathcal{Q}$ can be shipped via r distinct paths with disjoint peers to the data sources (including replicas) in the overlay network.*

[5]The replication process requires cooperation of neighboring peers, which is also assumed by other work in the literature [27]

[6]When physical peers are also located at intermediate levels (*e.g.,* in BATON), $\mathcal{Q}' = (start, start + 2 \times r \times \frac{D}{N})$ so that r peers at the leaf level are covered.

Proof. By evenly partition $\mathcal{Q}' = (start, start + r \times \frac{D}{N})$ into r sub-queries, the data results corresponding to $\mathcal{Q}'$ are expected to be located at r contiguous peers in the overlay network. Since each sub-query is issued by a distinct peer that is in $\mathcal{P}$'s horizon, it is expected that the sub-queries are shipped via r distinct paths between interlacing peers (*i.e.,* the sub-query issuing peer and the destination peer hosting the data results), according to Lemma 5. Denote by $\mathcal{P}'$ the destination peer that is responsible for the data results of query $\mathcal{Q}$. Since $\mathcal{P}'$ already replicates the results among all r peers in its horizon, while these r peers are expected to be the destination peers of the query routing of $\mathcal{Q}'$, the process to resolve the sub-queries of $\mathcal{Q}'$ is identical to the shipping of $\mathcal{Q}$ to the r peers in the horizon of $\mathcal{P}'$. During the process, r distinct paths with disjoint peers will be employed. Thus Theorem 6 holds. □

For example, the disjoint peer replication scheme is illustrated in Figure 3.9, where the horizon of peer $\mathcal{P}$ consists of three peers. Correspondingly three paths with disjoint peers (marked by nodes with the same color in Figure 3.9) are employed to ship query $\mathcal{Q}$ towards $\mathcal{P}'$ and the other peers in its horizon.

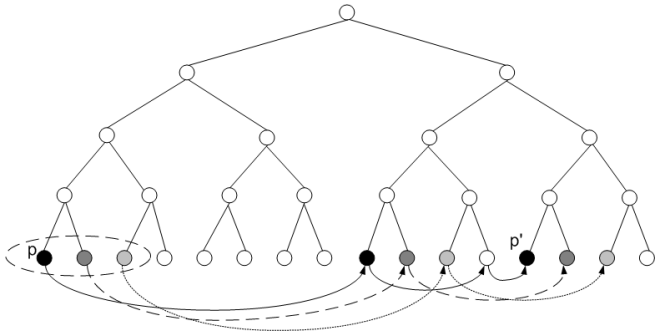


Figure 3.9: Query Shipping via Distinct Paths

Overlay Network Change and Routing Algorithm

The disjoint peer replication scheme requires each peer to explore and maintain the IP addresses of the peers in its horizon. However, this is conducted independently at each peer, without incurring overhead at the time of query processing. Moreover, the scheme does not change routing mechanisms in underlying overlay networks. For clarity, query processing enhanced with the disjoint peer replication scheme initiated at $\mathcal{P}$ is described in Algorithm 5.

Note that multiple (r) peers are expected to send back query results; thus multiple query result copies may be received by the query issuer. A duplicate removal process can be simply realized at the query issuer side. Moreover, due to data replication, any peer needs to propagate data updates to all replicas in the horizon. This may incur nontrivial maintenance overhead when the data update rate is high. Thus the proposed scheme is best suited for those applications that do not require frequent data updates.

Algorithm 5 *range_query_processing_disjoint_peer*($\mathcal{Q}$)

1: $\mathcal{P}$ relaxes $\mathcal{Q}$ to $\mathcal{Q}'$;
2: $\mathcal{P}$ evenly partitions $\mathcal{Q}'$ into r sub-queries and deploys them to the peers $\mathcal{P}_i$ in $\mathcal{P}$'s horizon;
3: **for** the query shipping starting from each peer $\mathcal{P}_i$ **do**
4: **if** the current peer $\mathcal{P}'$ contains the results matching the sub-query **then**
5: $\mathcal{P}'$ retrieves the replica matching $\mathcal{Q}$ and replies $\mathcal{P}$ with results;
6: **else**
7: $\mathcal{P}'$ ships the sub-query via the underlying routing protocol;
8: **end if**
9: **end for**

3.4 Performance Evaluation

To study the proposed proposal, the discrete event simulator p2psim[7] is used.

3.4.1 Evaluation of VOILA

A network topology is generated by uniform randomly mapping peers to coordinates in a square area with each side equivalent to 100 ms of delay, which is roughly the diameter of the Internet core today. The latency between two peers is then estimated by their Euclidean distances.

To simulate link failures, each hop in the overlay network fails with a specific rate p at the start of each epoch, after which queries are issued concurrently. Specifically, denote by N the network size; $p * N$ randomly chosen peers leave the overlay network at the start of each epoch; all the routing hops via these leaving peers will fail during the epoch. Each epoch lasts 15000 ms, which is referred to as *epoch time* in the remainder. To keep network size steady, new peers join the overlay network with the same rate. Networks consisting of up to 4000 peers are considered in this simulation. The query execution performance is averaged over 10 epochs. The average response time of each query is shorter than the period of an epoch, such that the effect of network churns (initiated at the starting of epochs) on query execution performance can be studied.

The range query load is synthesized as follows: the contiguous span τ of each range query follows Poisson distribution with mean $\frac{MAX}{N}$, where MAX is the max integer and N is network size. 1000 queries are generated during each epoch, where each query is issued by a uniform randomly chosen peer at a rate by following Poisson distribution (with mean $\frac{1000}{epoch\ time}$).

System failure rates are measured with different network churn rates (*i.e.,* $p = 10\%$ to 40%). Since failure rates are dominated by the duplication factor (*i.e.,* m), the metric for different duplication factors (*i.e.,* $m = 5$, $m = 10$ and $m = 15$) are plotted. As illustrated in Figure 3.10, the results show that a system with smaller m is more subject to failure under higher churn rates. The feasibility of the configuration of m is tested by measuring the metric for VOILA that adjusts m according to Theorem 2 in Section 3.2.1, as described next.

[7] p2psim:http://pdos.csail.mit.edu/p2psim/

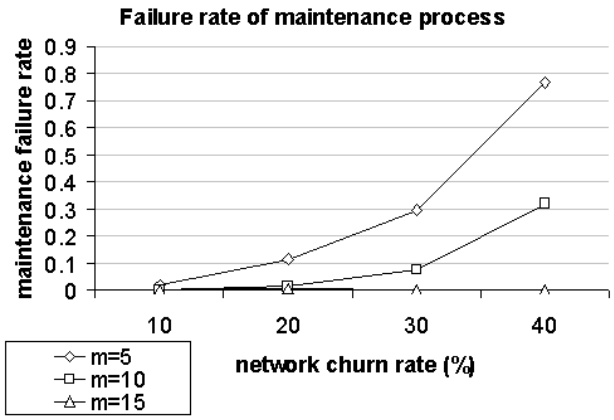


Figure 3.10: Robustness of VOILA

The average number of routing hops per query over VOILA with different network churn rates (*i.e.,* 10%, 20% and 30%) is evaluated. By fixing $b = 2$, VOILA with configured parameters (*i.e.,* l and m) is compared against that with fixed parameters ($l = 35$ and $m = 11$), which are computed according to Theorem 2 based on the setting when $p = 5\%$. As shown in Figure 3.11, the routing hops consumed by VOILA with configured parameters are consistently fewer, indicating that the appropriate choice of VOILA parameters effectively improves query execution performance.

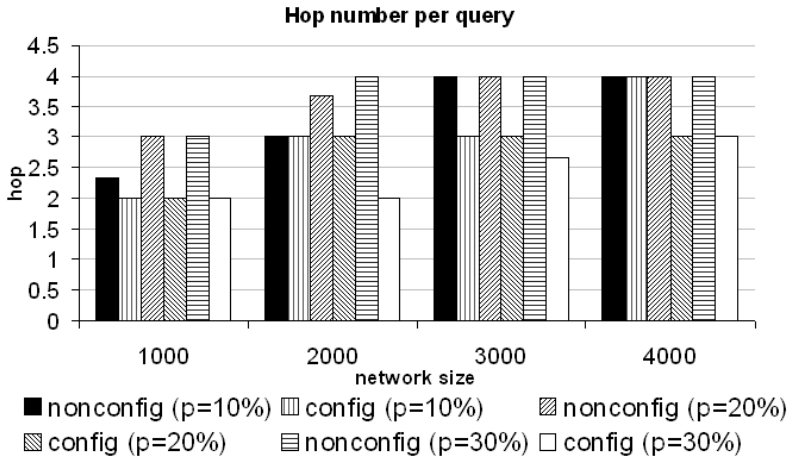
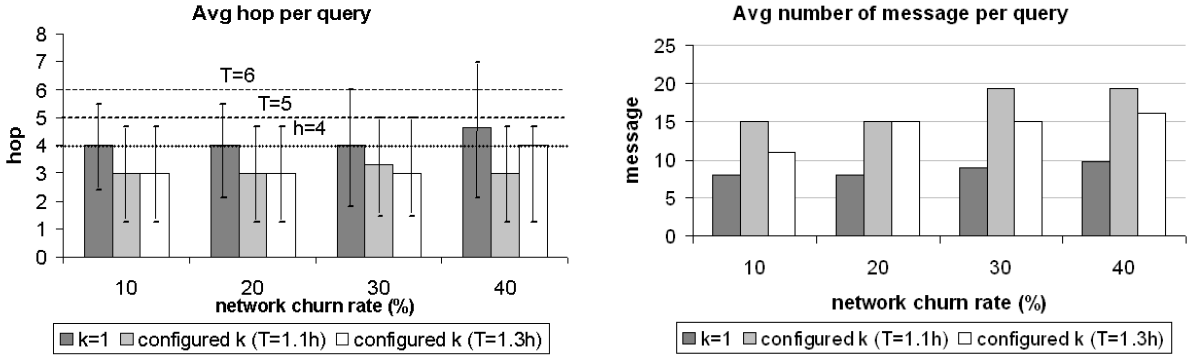


Figure 3.11: Query Performance of VOILA

The average latency per maintenance process for VOILA with configuration (of ρ) is measured with different network churn rates, which remains around 160 ms. This is because the maintenance cost is decided by the height of the tree overlay, which does not change significantly during network churn. Note that the average latency per maintenance process is proportional to the height of the VOILA tree, ranging between 2 and 3 under this evaluation setting; thus the latency of maintenance process is no more than 3 times of the average latency between peers, which is around 70 ms[8].

[8]The maximum latency equals to $\sqrt{2} * 100$ ms based on the simulation network setting.

To evaluate the effectiveness of the support of fault-tolerance specification with strong performance requirements (*i.e.,* T), the branching factor and duplication factor are fixed (*i.e.,* $b = 2$, $l = 35$ and $m = 11$) and the number of hops per query for VOILA with and without configuration of k is measured with a network consisting of 3000 peers. With selection of k, two fault-tolerance specifications with constraints $T = 5$ and $T = 6$ are considered, where $h = 4$ is the height of the tree overlay. Both the mean (illustrated by columns) and the standard deviation (illustrated by vertical bars) of the number of hops per query are presented in Figure 3.12(a). It is obvious that the number of hops per query for VOILA with a proper choice of k satisfies the T constraints. In contrast, when k is not configured (*i.e.,* $k = 1$), the system takes significantly more hops (even beyond T) to complete query processing, leading to performance failure. The corresponding bandwidth cost (measured by the number of messages) is illustrated in Figure 3.12(b), where the system that configures k consumes more bandwidth because it retries multiple links during each peer communication.



(a) (b)

Figure 3.12: Performance with and without Configuration of k

A sensitivity test on the impact of p over k to satisfy $T = log_b \frac{N}{l}$ is also conducted, where $b = 2$ and $N = 4000$. It shows that k is configured no larger than 7 when $p \leq 40\%$. Thus, if the range of network churn rate is known, and the optimization of bandwidth cost is not crucial, An upper bound of k can be computed and pre-installed as a default value.

3.4.2 Evaluation of Disjoint Link Replication Scheme

With different network churn rates (*i.e.,* link failure rate $p = 10\%$ to 50%), the number of hops per query of a BATON-style overlay network is measured with and without the deployment of the disjoint link replication scheme.

The fault-tolerance specification constraint is set to be $T = 6 \approx \frac{log_2(4000)}{2}$, which is the expected number of hops per query under static network settings. The evaluation results in Table 3.1 show the ratio of the performance failures of 1000 range queries when the replication factor $r = 1$ (*i.e.,* without deployment of replication strategy) and r values computed with different $j \in [1, 6]$). It is demonstrated that the disjoint link replication scheme improves fault-tolerance.

settings \ p	0.1	0.2	0.3	0.4	0.5
$j = 1$	0	0	0	0	0
$j = 2$	0	0	0	0	0
$j = 3$	0	0	0	0.05%	0.05%
$j = 4$	0	0	0	0	0
$j = 5$	0	0	0	0	0
$j = 6$	0	0	0	0	0
non-config	0.25%	0.65%	1.6%	5.2%	12.3%

Table 3.1: Performance Failure Ratio of Disjoint Link Replication Scheme

The number of hops per query (both mean and standard deviation) is measured under the same setting. The results are shown in Figure 3.13. With the disjoint link replication scheme, query shipping cost is significantly decreased. For simplicity, those queries with performance failure are terminated when the number of hops goes beyond $log_2(4000) \approx 12$. Thus the actual cost of the "non-config" setting may potentially be higher.

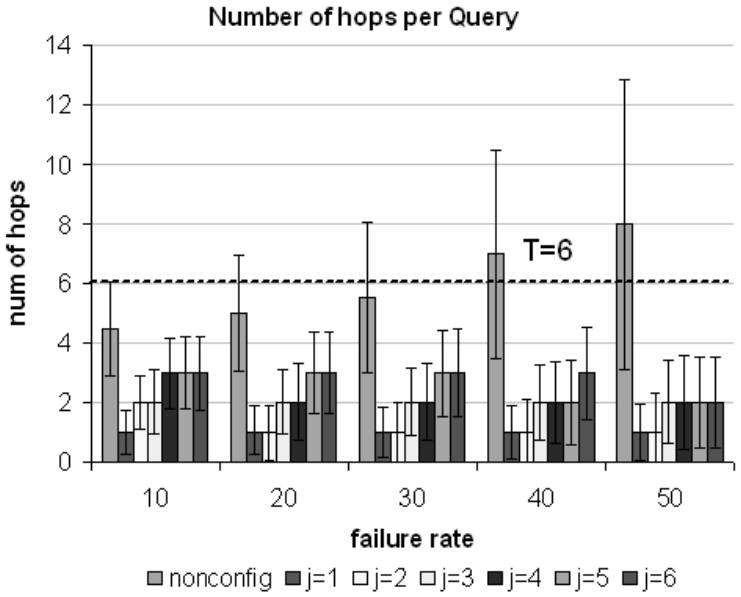


Figure 3.13: Performance with and without the Disjoint Link Replication Scheme

An overhead of the proposed approach is the redundant number of messages since each query is replicated via multiple paths. However, because queries are usually small (*e.g.,* only containing a range constraint), the overall bandwidth consumption is not significant. In addition to the random sampling of each peer, uniform gossip mechanism is also effective in propagating range queries. However, it is expected to incur a larger number ($\mathcal{O}(N \ln N)$) of messages than that incurred by the proposed approach, which is ($\mathcal{O}(\frac{rNj!}{log^j \frac{N}{2}})$) where N is network size and j is a small constant (*e.g.,*

$j \leq 6$). Moreover, with gossip mechanism, all peers have to be synchronized in the propagation process, while in the proposed approach, only $\mathcal{O}(\frac{rNj!}{log^j \frac{N}{2}})$ are contacted for the sampling process, reducing the coordination costs among peers.

3.4.3 Evaluation of Disjoint Peer Replication Scheme

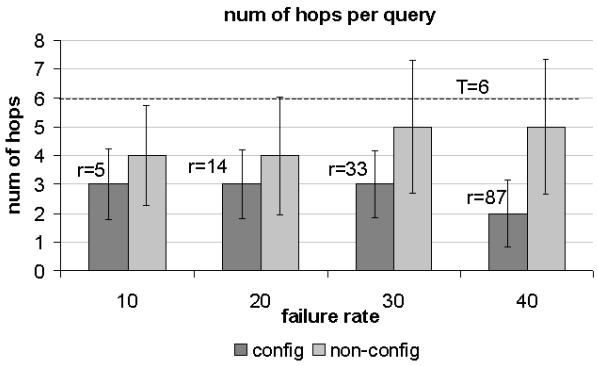


Figure 3.14: Query Shipping Cost with and without the Disjoint Peer Replication Scheme

To study the disjoint peer replication scheme, the number of hops per query (including mean and standard deviation) is evaluated under different peer failure rates (*i.e.,* $p = 10\%$ to 40%[9]) when $T = 6$. The average numbers of hops per query of "config" (with respect to the disjoint peer replication scheme) and "non-config" settings are shown in Figure 3.14, where the values of the replication factor r are illustrated as well. The performance failure ratio is presented in Table 3.2. Since distinct paths with disjoint peers are employed in query shipping, the number of hops is expected to satisfy fault-tolerance specifications (with an ignorable performance failure ratio).

settings \ p	0.1	0.2	0.3	0.4
config	0.1%	0.2%	0.2%	0.1%
non-config	2%	4.5%	7%	7%

Table 3.2: Performance Failure Ratio of Disjoint Peer Replication Scheme

Finally, the overhead of the disjoint peer replication schemes is incurred due to redundancy in query shipping. The overall bandwidth consumption increases linearly proportional to the number of paths employed for query shipping, which may not be significant since queries are usually small. Query processing latency does not increase simply because query shipping via multiple paths is conducted concurrently.

The experimental results demonstrate that, by using the disjoint peer replication scheme, performance failure can be effectively reduced even under high network churn, improving the fault-tolerance of the system and guaranteeing query execution performance.

[9]Note that here failure rates are with respect to peers rather than links, as considered when the disjoint link replication scheme is evaluated.

3.5 Summary

Tree-based overlay networks are widely studied to support range query processing in large-scale P2P networks. However, existing systems lack the capability to configure query execution performance according to network churn rates, leading to performance failure with respect to fault-tolerance specifications that require strong performance guarantees.

To solve this problem, a highly configurable tree-based overlay network, called VOILA, is developed, which allows the configuration of both overlay network structure and routing algorithm based on network churn rates, supplying strong performance guarantees. Moreover, for existing overlay networks without such configurability, effective replication schemes are proposed to enhance their configurability at the query processing level, such that the fault-tolerance specifications can be satisfied.

Extensive simulations show that the proposed approaches effectively enhance system fault-tolerance against link and peer failures and provide strong performance guarantees with respect to the required fault-tolerance specifications under varying network churn rates.

Chapter 4

Range Query Processing Scheme for Unstructured P2P Networks

Under unstructured P2P architecture, data are kept on hosted peers and usually no distributed indexes are built for query shipping. When range queries are propagated to a peer, the peer can match its data against the range constraints locally without communication with other peers. The propagation of queries is realized through flooding [99], gossip [56] or random walk [41] mechanisms. However, when multiple range queries are considered and many of them are identical (*e.g.,* issued by different peers), caching is a rule-of-thumb design principle that has been employed under existing unstructured P2P architecture, which reuses the results obtained from previous query processing. Different from the caching mechanisms that support point queries, range queries may request multiple data items, *some of which may not be well cached (referred to as "poorly-replicated " data items) such that the overall range query processing is dominated by the retrieval of these data items.*

This dissertation studies the distributed range caching problem under unstructured P2P architecture. In particular, purely decentralized mechanisms are developed to locate the poorly-replicated data items that are potentially queried in the future, and an adaptive prefetch-based caching approach is proposed to improve the performance of the range query processing that involves poorly-replicated data items. The effectiveness of the approach is demonstrated theoretically by proving that, under a specific query distribution model with an increasing query load, the approach can improve overall performance of the query processing that retrieves poorly-replicated data items by at least a factor of $\mathcal{O}(\ln m)$, where m is the number of queries, even when network churn and cache expiration exist. Through extensive simulations, the effectiveness of the proposed approach is also demonstrated under various other query load settings.

The organization of the remainder of this chapter is as follows. Section 4.1 gives an overview of the design principles of the proposed approach. In Section 4.2 the performance of cache-based range query processing is analyzed in unstructured P2P overlay networks. Section 4.3 covers the prefetch-based approach that results in substantial communication cost savings for range query processing. Performance evaluation results are presented in Section 4.4. The chapter is concluded in Section 4.5.

4.1 Overview of Design Principles

Without loss of generality, consider using random walk mechanism for query shipping, while the following discussion is applicable for flooding and gossip-based mechanisms as well. Since data are cached at peers after retrieval, subsequent queries for the same data can be answered by multiple caches, facilitating the search process. In the case of range query processing, the queries can also be shipped within the network through flooding. Data retrieved after the query execution are easily cached at query issuers, which produce *distributed range caches* that can potentially be used during subsequent range query processing.

Because unstructured P2P overlay networks are built without the knowledge of data placement, average communication cost (*e.g.,* the number of messages or latency) per query is inversely proportional to the number of data copies, or replicas, in the network that satisfy the query [23]. Those data results that are not well-replicated may incur a higher cost to retrieve, delaying the progress of the range query processing. For simplicity, all peers are assumed to be sufficiently powerful to keep data replicas and process related range queries. In practice, peers with heterogeneous storage and processing capacity may act as super-peers. Such a general heterogeneous model is not studied in this dissertation. However, in the proposed approach, peers may impose different constraints on cache storage capacity, which will affect the computation of the data correlation parameter τ, as detailed in Section 4.3.1.

It is often the case that range query results include data items that are not well replicated. For example, in a location-based hotel reservation system in P2P networks, "popular" hotels, such as those close to a conference site, are usually queried first by conference participants. When these hotels are booked in full, users tend to relax the range constraints (*e.g.,* with respect to geographical proximity) to explore other hotel information, which may not be well replicated in the network yet. For simplicity, these data items that are not well replicated are referred to as "poorly-replicated " data items. The approach proposed in this work will predict and prefetch those poorly-replicated data items that may potentially be requested in subsequent range queries and then facilitate the caching of these data to improve overall query execution performance.

Existing approaches are not effective in facilitating the caching of the poorly-replicated data items that may potentially be queried in the future. In unstructured P2P overlay networks, *uniform replication scheme* (deployed in KaZaa) caches each data item at a fixed number of peers so that it preventively excludes poorly-replicated data items from the system. However, this scheme is oblivious to the knowledge of query distribution such that the fixed number of peers that manage the data items covered by "popular" queries may be overloaded. Moreover, the replication scheme requires strong altruistic cooperation from peers in that peers may cache data items regardless of whether they are issuing queries, which may not be feasible in uncooperative P2P environments. In contrast, with the *proportional replication scheme* that is employed in Gnutella, each peer only caches results after query execution such that caching process is triggered by query processing and only query issuers rather than arbitrary peers cache the results. Similarly, *square-root replication scheme* [23] makes the number of cached data items proportional to the square root of the number

of the corresponding queries, improving average query processing performance with respect to constrained cache sizes. In comparison to the uniform replication scheme, the last two schemes achieve better load-balancing by considering query distributions and do not rely on altruism from peers. However, they disregard the caching of poorly-replicated data items, which may bottleneck the performance of the range query since these data are part of the range query result.

The key to the problem is to recognize those poorly-replicated data items that may potentially be requested by range queries in the future, and to facilitate the caching of these data items. With respect to unstructured P2P overlay networks, the following design principles are considered: (1) the approach needs to be purely decentralized; (2) the approach should be efficient, without incurring significant communication or computational overhead; and (3) it is desirable that the approach can be deployed with existing routing protocols and replication schemes that have been developed for unstructured P2P networks.

Using the intuition that well-replicated data are "popular" data cached in the P2P system, In this work, a popularity-aware *prefetch-based* approach is proposed to facilitate the caching of poorly-replicated data items. Prefetching in the context of other application domains (*e.g.,* compiler technology) has been well-studied. the proposed approach, in the context of prefetching range data in P2P systems, is novel in that it is data-popularity aware: (1) peers independently collect global information about the relationship between poorly-replicated data items and "popular", well-replicated, ones involved in previously executed queries, enabling an adaptive approach for range query processing (see Section 4.3 for details); (2) since popular data are easily obtained from peers through flooding or random walk mechanism, simply prefetching poorly-replicated data items can be more cost-effective with respect to bandwidth consumption; and (3) sufficient query issuers will exist over "popular" data items, providing opportunities to piggyback "correlated" poorly-replicated data items onto popular data and thereby facilitating the prefetching process.

4.2 Cache-based Range Query Processing

Range queries typically involve the range constraints that are defined over numeric-valued data [80]. The processing of a specific range query completes when all distinct data items (either from data sources or caches) that satisfy the range constraints have been retrieved. Range queries can be issued by any peer in the system, and be shipped to other peers through flooding, gossip, or random walk routing mechanisms. Since flooding is no more effective than random walk with respect to routing cost [23], while gossip mechanism is based on random walk in that queries are shipped to randomly chosen peers in the network, random walk mechanism is employed in this work without loss of generality.

Based on the random walk mechanism, both the number of messages and the latency to retrieve a specific data item are inversely proportional to the number of the data item replicas in the network. Suppose that the cost function (denoted by $Cost$) of range query processing is defined over $\mathcal{Q} \times R$, where $\mathcal{Q}$ denotes the set of range queries and R is the real value domain of query processing cost[1].

[1]In this work, the communication costs regarding the number of messages and query processing latency are

Given a query $q \in \mathcal{Q}$, $Cost(q) \propto \frac{1}{r_q}$ (*i.e.,* $Cost(q)$ is proportional to $\frac{1}{r_q}$), where r_q is the lower-bound of the number of data item replicas that can satisfy q. When there exist multiple distinct data items $\{s_1, s_2, ...\}$ in the network that satisfy range query q, $r_q = min(|s_1|, |s_2|, ..., |s_i|, ...)$, where $|s_i|$ denotes the corresponding numbers of data item s_i replicas.

Obviously, the overall performance of range query processing is affected by the involved poorly-replicated data items. The period that specific data items have not been well replicated is referred to as the *cold period*. Suppose that the results of a range query q include data item s, which initially has a single replica (*i.e.,* the original data item itself) in the network; when there are m subsequent q queries issued, the overall query execution cost during s' cold period is computed as below, based on the well-established proportional and square-root replication schemes respectively. For simplicity, each execution of a certain query is initiated by a distinct peer.

- *Proportional replication scheme* Each query execution is expected to increase the number of replicas by one, such that the overall query processing cost with respect to m q queries, denoted by $Cost(q)$, is derived as below. Because sequence 4.3 does not converge, an approximate result is presented with respect to a considerably large m.

$$\begin{aligned} Cost(q) &= \Sigma_{i=1}^{m}(\frac{N}{i}) && (4.1)\\ &= N + \frac{N}{2} + \frac{N}{3} + ... + \frac{N}{m} && (4.2)\\ &\approx N \times \ln m && (4.3) \end{aligned}$$

- *Square-root replication scheme* Although the square-root replication scheme performs better than the proportional replication scheme with respect to range query processing performance under constrained overall cache sizes [23], the range query processing involving poorly-replicated data items may incur higher communication cost. The following derivation presents the overall query processing cost, where sequence 4.5 increases monotonically and never converges. For simplicity, the number of replicas exactly equals the square root of the corresponding number of queries[2], which does not affect the validity of the obtained result.

$$\begin{aligned} Cost(q) &= \Sigma_{i=1}^{m}(\frac{N}{\sqrt{i}}) && (4.4)\\ &= N + \frac{N}{\sqrt{2}} + \frac{N}{\sqrt{3}} + ... + \frac{N}{\sqrt{m}} && (4.5) \end{aligned}$$

The above analysis shows that range query processing performance is affected when queries retrieve poorly-replicated data items during their cold period. In the next section, the prefetch-based approach is proposed, where poorly-replicated data items are prefetched by query issuers that

considered.

[2]The actual number of replicas equals the square root of the corresponding query load size multiplied by a constant factor [23].

request the well-replicated data items "correlated" to the poorly-replicated ones. This potentially decreases the retrieval cost of the poorly-replicated data items within cold periods and improves the range query execution performance.

4.3 Prefetch-based Caching

In this section, data *correlation* is addressed first, which materializes the locality concept that is essential to prefetch-based mechanisms [86]. Then the design details of the prefetch-based caching approach are presented.

4.3.1 Data Correlation

To quantify the correlation between poorly-replicated data items and well-replicated ones, a distance function D is introduced. For Euclidean range space, when data items represent point data (*e.g.,* the longitude and latitude information of locations), D can simply be the one-dimensional Euclidean distance function between the point values[3]. Instead, when data items correspond to range segments (*e.g.,* the range span of longitude and latitude information of a region around a specific location), D may be defined over the Euclidean distance between the centroid points (*e.g.,* median points in one-dimensional space) of the corresponding range segments. Other applications may employ their customized distance functions, which does not affect the applicability of the prefetch-based approach.

Based on the distance function D, data *correlation* is defined as follows: *two data items s and s' are correlated if $D(s, s') \leq \tau$*. The correlation threshold τ can be pre-defined and configured by peers when they join the overlay network, which may not be sufficiently flexible since the threshold may be over or under-valued. For example, with respect to a specific range query load, when τ is set too low, data items that are covered by the same range queries may potentially be regarded uncorrelated; in contrast, when τ is set too high, more irrelevant data items may be regarded correlated even if they are never queried together. Since poorly-replicated data items are piggybacked with the well-replicated query results, τ is measured based on the distances between poorly-replicated data items and well-replicated ones from the history of executed range queries.

On one hand, this approach takes the information of range queries into account such that: (1) the inherent correlation of data items within the same range query is captured; and (2) since intuitively the well-replicated data items are "popular" among peers, their correlated (poorly-replicated) data items may also become "popular" with a higher probability, which is recognized by the proposed approach. On the other hand, the approach is popularity-aware, enabling *adaptive* data prefetching: the greater the portion of range query workload that retrieves poorly-replicated data items, the more precisely τ captures the expected distances between poorly-replicated data items and well-replicated ones. Conversely, when range queries seldom retrieve poorly-replicated data items, τ tends to be close to zero such that prefetching may not even be triggered.

[3]This does not conflict with the focus on range query processing since range queries may include multiple point values.

While there do exist correlations between poorly-replicated data items, or between well-replicated data items, these are not considered in this approach because: (1) the proposed approach relies on query issuers requesting well-replicated data items to prefetch poorly-replicated ones; the correlation between poorly-replicated data items is less important because it does not indicate the involved data items will be requested in subsequent queries; and (2) the retrieval of well-replicated data items incurs low processing cost, such that the prefetching of these data items may not be cost-effective with respect to bandwidth consumption. Performance evaluation (Section 4.4) supports the belief that simply prefetching all correlated data items regardless of data popularity may not be efficient.

Specifically, each peer records the history of the range queries that it issued previously. Each history record contains the maximum distance between any poorly-replicated data items and well-replicated ones that are involved in the range query at query execution time. Each peer then randomly samples a configurable number of other peers in the overlay network and obtains their query processing history records. This captures approximate global information about how poorly-replicated data items affect range query execution. The random sampling in unstructured P2P overlay networks can be realized through existing techniques [41, 54]. Query processing history records are collected by peers periodically such that they learn up-to-date knowledge on executed range queries.

For example, as shown in Figure 4.1, peer $\mathcal{P}_1$ randomly samples a number of peers (*i.e.,* $\mathcal{P}_2$, $\mathcal{P}_3$, $\mathcal{P}_4$ and $\mathcal{P}_5$) from the network and obtain their history records.

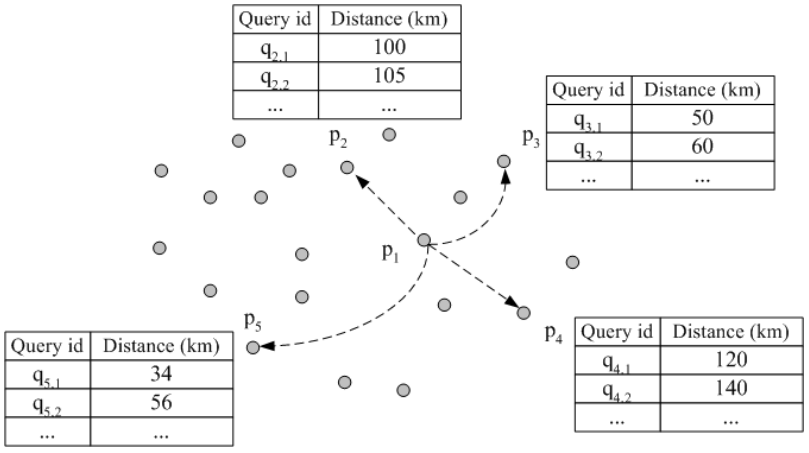


Figure 4.1: Sampling-based Estimation of τ

Once a set of distance values from the history records, denoted by $\{d_i\}$, are obtained, each peer $\mathcal{P}$ estimates the distribution of the distance values through kernel estimation technique [85], which is a non-parametric data distribution modeling scheme. Non-parametric modeling schemes are advantageous in P2P environments since no a-priori knowledge about data distribution is required. Then the expected distance value is chosen as τ. In this work, the commonly-used Gaussian kernel function $K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$ is employed, and the estimated probability density function (PDF) is $PDF_k(x) = \frac{1}{Sh} \Sigma_{i=1}^{S} K(\frac{x-x_i}{h})$, where S denotes the number of history records, x_i denotes the

distance value corresponding to the i_{th} sample, and h is a smoothing factor that is configurable with respect to specific applications. The derivation of τ is shown below, where $y_i = \frac{x-x_i}{h}$ for each specific i.

$$\begin{aligned}
\tau &= \int_0^\infty x \times PDF_k(x)dx = \int_0^\infty x \times \frac{1}{Sh}\Sigma_{i=1}^{S} K(\frac{x-x_i}{h})dx && (4.6)\\
&= \frac{1}{Sh\sqrt{2\pi}} \times \Sigma_{i=1}^{S}(\int_0^\infty x \times e^{-\frac{1}{2}(\frac{x-x_i}{h})^2}dx) && (4.7)\\
&= \frac{1}{Sh\sqrt{2\pi}} \times \Sigma_{i=1}^{S}(\int_0^\infty x \times e^{-\frac{1}{2}y_i^2}dx) && (4.8)\\
& && (4.9)
\end{aligned}$$

Define $y_i = x - x_i$,

$$\begin{aligned}
\tau &= \frac{1}{Sh\sqrt{2\pi}} \times \Sigma_{i=1}^{S}(\int_0^\infty x \times e^{-\frac{1}{2}y_i^2} \times hdy_i) && (4.10)\\
&= \frac{1}{S\sqrt{2\pi}} \times \Sigma_{i=1}^{S}(\int_0^\infty (hy_i + x_i) \times e^{-\frac{1}{2}y_i^2}dy_i) && (4.11)\\
&= \frac{1}{S\sqrt{2\pi}} \times \Sigma_{i=1}^{S}(\int_0^\infty hy_i \times e^{-\frac{1}{2}y_i^2}dy_i + \int_0^\infty x_i \times e^{-\frac{1}{2}y_i^2}dy_i) && (4.12)\\
&= \frac{1}{S\sqrt{2\pi}} \times \Sigma_{i=1}^{S}(h \times \int_0^\infty y_i \times e^{-\frac{1}{2}y_i^2}dy_i + x_i \times \int_0^\infty e^{-\frac{1}{2}y_i^2}dy_i) && (4.13)\\
&= \frac{1}{S\sqrt{2\pi}} \times \Sigma_{i=1}^{S}(h + x_i \times \frac{\sqrt{2\pi}}{2}) && (4.14)\\
&= \frac{h}{\sqrt{2\pi}} + \frac{1}{2S} \times \Sigma_{i=1}^{S} x_i && (4.15)
\end{aligned}$$

Observe that the computed expected value of τ is only slightly different from the average (*i.e.,* $\frac{\Sigma_{i=1}^{S} x_i}{S}$) of the sampled distant values. An engineering approach is to replace the expected τ computed through kernel estimation with the average to ease local computation costs. However, this may decrease the accuracy of the data correlation computation, potentially increasing the volume of prefetched data items that are irrelevant to subsequent queries.

With respect to range query processing, since query issuers and peers holding query results may obtain different values of τ, the query issuer's τ is used as the correlation threshold, because they can flexibly adjust the value of τ to include other constraints (*e.g.,* local storage constraint). For example, when the query issuer has a storage constraint, denoted by *sc*, it always sets $\tau = min(correlation\ threshold, sc)$ such that the local storage constraint is never violated.

4.3.2 Data Popularity

The computation of τ requires the knowledge of data *popularity*, which is used to distinguish poorly-replicated data items from well-replicated ones. Although the obtained query history records can be employed to estimate data popularity [102], they may not be sufficiently accurate to reflect the

overall data distribution in the overlay network. Thus, a purely decentralized approach is employed to estimate data popularity directly. Each peer evaluates the popularity of the local data through an exploration process. Consider a peer $\mathcal{P}$ with a set of data items, denoted by $\mathcal{S} = \{s_1, s_2, ...\}$. $\mathcal{P}$ issues an "exploration query" over each $s_i \in \mathcal{S}$, where a configurable Time-to-Live (TTL) counter is attached to each exploration query. During each hop, the TTL counter decreases by one and all appearances of s_i are recorded by $\mathcal{P}$. The shipping of an exploration query terminates when TTL equals zero. Once the exploration process completes for all data items, $\mathcal{P}$ figures out the number of cached replicas for each s_i during the exploration process. Those s_i with more replicas (*i.e.,* above threshold T) are regarded as "well-replicated " and the others are considered "poorly-replicated ". A similar approach has been employed in PIER to decide data popularity [47]. Both the TTL and T are configurable, where the TTL value is usually set to be small to avoid large explorations.

4.3.3 Prefetch-based Approach

Suppose that peer $\mathcal{P}$ receives a range query issued by $\mathcal{P}'$ that covers a data item s; $\mathcal{P}$ will reply to $\mathcal{P}'$ with all its cached data items $\{s_1, s_2, ..., s_i, ...\}$ that satisfy the following conditions: (1) each s_i is correlated to s based on the distance function D and the correlation threshold τ that is attached to the query (*i.e.,* $D(s, s_i) \leq \tau$); and (2) based on the data popularity measurement of $\mathcal{P}'$ (*i.e.,* the query issuer), s is well-replicated and s_i is poorly-replicated.

For instance, consider multiple query issuers requesting data items s and s' through queries q_1 and q_2 respectively, as illustrated by white and grey nodes in Figure 4.2(a). For simplicity, suppose that s and s' are cached at peer $\mathcal{P}$ that receives the queries. Then Figure 4.2(b) demonstrates the proportional replication scheme and Figure 4.2(c) shows the proportional replication scheme enhanced with prefetching, where peers denoted by dark nodes eventually cache the poorly-replicated data item s'. Due to the prefetching of s' by query issuers over q_1, s' is cached more quickly in Figure 4.2(c).

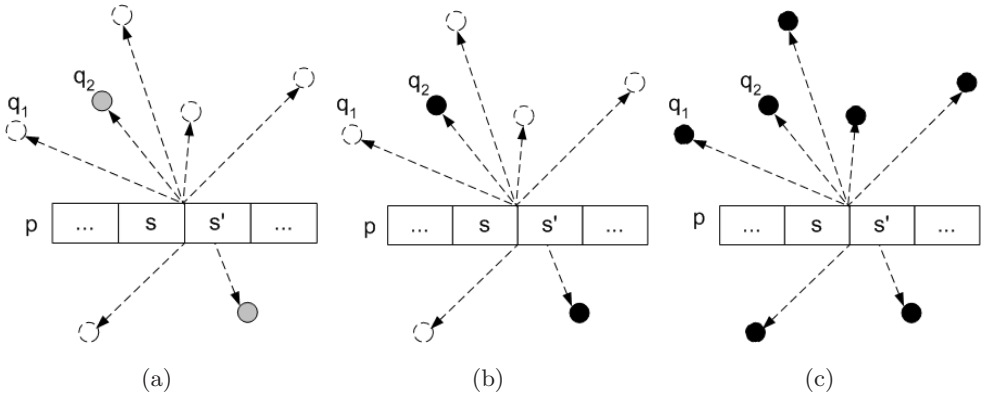


Figure 4.2: Caching Approaches with and without Prefetching

The prefetch-based approach is presented in Algorithm 6 regarding peer $\mathcal{P}$ when it receives a query q covering data item s that is issued at $\mathcal{P}'$. The approach has several advantages: (1) it makes slight changes to existing replication schemes without adjusting the overlay network architecture

or routing algorithms; all decisions, including the computation of the data correlation threshold and data popularity, are made independently without costly coordination among peers; (2) poorly-replicated data items are piggybacked during the query processing over well-replicated data items, saving communication cost; moreover, peers are not required to be altruistic since only query issuers who utilize the query processing services take on the prefetching overhead cost; and (3) the approach is purely decentralized without relying on any centralized mechanisms, thus providing scalability.

Algorithm 6 $prefetch\_based\_caching(q, \tau)$

```
1: if P learns that s is well cached then
2:   for all data items s' that are poorly-replicated and D(s, s') ≤ τ do
3:     if D(s, s') ≤ τ then
4:       P replies to P' the data item s;
5:     end if
6:   end for
7: end if
```

4.3.4 Guarantees on Performance Improvement

With the prefetch-based approach, the population of poorly-replicated data items is affected by the query volume over the correlated well-replicated data items. Under the following query distribution, the proposed approach guarantees that the overall range query cost over poorly-replicated data items is $N \times \mathcal{O}(1)$, which is at least $\mathcal{O}(\ln m)$ factor less than the counterpart when no prefetching is enforced.

Consider a range query q_1 involving data item s and query q_2 involving data item s', where s is well-replicated while s' is poorly-replicated. Suppose that, initially only one replica of s' exists in the network, while there exist n replicas of s including original and cached ones. Consider a query load consisting of m_1 q_1 queries and m_2 q_2 queries. For simplicity, all q_1 (and q_2) queries are uniform-randomly distributed across a certain period of time; thus it is expected that $\frac{m_1}{m_2}$ q_1 queries are processed when each q_2 is processed. Note that this assumption is not necessary for the following analysis; it will be clear shortly that only if the number of q_1 queries is sufficiently large compared with that of q_2 queries, the analysis will hold. The number (n) of s replicas is also assumed to be stable, and the case when n may change will be discussed shortly. It is proven that Theorem 7 holds, where $\delta > 1$ is a system parameter that affects the constant factor of the average query operation cost (*i.e.,* $\mathcal{O}(1)$), and N denotes the network size. A proof is presented subsequently.

Theorem 7. *When* $\frac{m_1}{m_2 \times n} \geq 2^{\delta}$*, the overall cost for processing* m_2 q_2 *queries equals* $\mathcal{O}(1) \times N$*.*

Proof. Suppose that initially, there is one replica of s' and it is cached together with s on a peer. This does not affect the generality of the analysis since subsequently prefetched s' will be cached together with s. During each period, one q_2 is processed and $\frac{m_1}{m_2}$ q_1 queries are issued such that the expected number of q_1 queries that visit a peer holding s' equals $\frac{m_1}{m_2 \times n}$. When $\frac{m_1}{m_2 \times n} \geq 2^{\delta}$, the

number of cached s' data item copies increases by 2^δ fold. Recursively, suppose $i > 1$, when there are $(i-1)^\delta$ s' replicas in the network after the execution of the $(i-1)_{st}$ q_2 query, the expected number of peers that cache s' through the execution of query q_1 equals $\frac{m_1}{m_2 \times n} \times (i-1)^\delta$ and is no less than i^δ, as shown in the following derivation.

$$\frac{m_1}{m_2 \times n} \times (i-1)^\delta > (2 \times (i-1))^\delta \quad (4.16)$$

$$\geq ((1 + \frac{1}{i-1}) \times (i-1))^\delta = i^\delta \quad (4.17)$$

Consequently, when the execution of the i_{th} q_2 query completes, no less than i^δ peers will cache s'. The overall shipping cost to resolve all q_2 queries is then computed as follows. While q_2 queries are executed sequentially, the above analysis is easily extended to handle concurrent query execution.

$$Cost(q_2) \leq \Sigma_{i=1}^{m_2} \frac{N}{i^\delta} \quad (4.18)$$

$$= N + \frac{N}{2^\delta} + \frac{N}{3^\delta} + ... + \frac{N}{m_2^\delta} \quad (4.19)$$

Since sequence 4.19 is a Riemann-Zeta sequence [34], it converges for all real values $\delta > 1$. Thus the overall query operation performance with respect to the query load of q_2 is $Cost(q_2) = \mathcal{O}(1) \times N$ when m_2 is considerably large. □

For example, when $\delta = 1.5$, $Cost(q_2) \approx 2.612 \times N$. This overall cost is at least $\mathcal{O}(\ln m)$ times less than those achieved by existing replication schemes (addressed in Section 4.2) that do not employ prefetching.

A sufficient condition of Theorem 7 is that the ratio of the number of q_1 queries over the number of s caches (denoted by n) is no less than 2^δ. However, multiple factors may affect this number in practice: (1) after the execution of q_1 queries, peers are capable of caching s, increasing n; (2) under network churn, peers may fail (or leave) suddenly, decreasing n; and (3) when cache expiry schemes are deployed in P2P systems for data freshness [10, 50], n may also change.

Cache replacement may affect the value of n as well. Specifically, if the cache size is sufficiently large, peers can hold all recently cached data items and cache replacement would happen infrequently. If the cache size is limited, cache replacement behavior can be easily integrated with cache expiry. For example, a widely employed recency-based cache replacement strategy such as Least-Recently-Used can be enforced through the cache expiry process by choosing an appropriate expiry period. This would evict least-recently-used data items from caches after each expiry period.

Theorem 8 proves that, a certain q_1 query load can satisfy the sufficient condition of Theorem 7 even when network churn and/or cache expiry occur. Recall that, during each period, one q_2 query is issued. Any peer (and cache) fails (or leaves) with a probability of $0 \leq r \leq 1$ per period, and all cached data items expire after $k > 0$ periods of time. $l(i)$ represents the expected number of q_1 queries during the i_{th} period, $n(i)$ denotes the number of s data item replicas after the i_{th} period, and $n(0) = n$ denotes the initial number of s replicas.

Theorem 8. *When the query load of q_1 satisfies the following distribution,*

$$l(i) = \begin{cases} 2^{\delta+1}(1+2^{\delta+1})^{i-1}(1-r)^{i-1}n, & when\ i < k \\ 2^{\delta+1}(1+2^{\delta+1})^{i-1}(1-r)^{i-1}n - \mathcal{O}((1+2^{\delta+1})^{i-k}(1-r)^{i})), & when\ i \geq k \end{cases}$$

the number of s' replicas will be no less than i^{δ} after the i_{th} period, where $\delta > 1$; consequently, the overall cost of processing m_2 q_2 queries is bounded by $\mathcal{O}(1) \times N$, even under network churn and cache expiry.

Proof. For brevity, the case when $i \geq k$ is proved. It is easy to apply the same derivation for the case when $i < k$, where no cached data expires.

When $i \geq k$, the number of s replicas (denoted by $n(i)$) after the i_{th} period is computed below, where the first component consists of the accumulated number of s until the $(i-1)_{st}$ period, added by the increase of the s (denoted by $l(i)$) during this period; the second component covers the loss of s replicas due to cache expiry. Both components are multiplied by corresponding damping factors to reflect network churn.

$$\begin{cases} n(i) = (n(i-1) + l(i)) \times (1-r), & when\ i < k \\ n(i) = (n(i-1) + l(i)) \times (1-r) - l(i-k) \times (1-r)^k, & when\ i \geq k \end{cases}$$

Consider a more relaxed function $n'(i) = (n'(i-1) + l(i)) \times (1-r) - (1-r)^k$, where $n'(0) = n(0) = n$. Since $(1-r)^k \leq l(i-k) \times (1-r)^k$, it is obvious that $n'(i) \geq n(i)$. Then similarly consider,

$$\begin{cases} n'(i) = (n'(i-1) + l(i)) \times (1-r), & when\ i < k \\ n'(i) = (n'(i-1) + l(i)) \times (1-r) - (1-r)^k, & when\ i \geq k \end{cases}$$

Suppose $A = 2^{\delta+1}$ and make $l(i) = A \times n'(i-1)$. The following derivation computes $n'(i-1)$.

$$\begin{aligned} n'(i-1) &= (n'(i-2) + l(i-1)) \times (1-r) - (1-r)^k \\ &= (1+A) \times n'(i-2) \times (1-r) - (1-r)^k \\ &= (1+A) \times ((1+A) \times n'(i-3) \times (1-r) - (1-r)^k) \times (1-r) - (1-r)^k \\ &= (1+A)^2 \times n'(i-3) \times (1-r)^2 - (1+A) \times (1-r)^{k+1} - (1-r)^k \\ &= ... \\ &= (1+A)^{i-k-1} \times n'(k) \times (1-r)^{i-k-1} - \Sigma_{x=1}^{i-k-2}((1+A)^x(1-r)^{x+k}) \\ &= (1+A)^{i-k} \times n'(k-1) \times (1-r)^{i-k} - \Sigma_{x=1}^{i-k-1}((1+A)^x(1-r)^{x+k}) \\ &= (1+A)^{i-k+1} \times n'(k-2) \times (1-r)^{i-k+1} - \Sigma_{x=1}^{i-k-1}((1+A)^x(1-r)^{x+k}) \\ &= ... \\ &= (1+A)^{i-1} \times n'(0) \times (1-r)^{i-1} - \Sigma_{x=1}^{i-k-1}((1+A)^x(1-r)^{x+k}) \\ &= (1+A)^{i-1} \times n \times (1-r)^{i-1} - \Sigma_{x=1}^{i-k-1}((1+A)^x(1-r)^{x+k}) \\ &= [(1+A)(1-r)]^{i-1} \times n - \frac{(1+A)^{i-k}(1-r)^i}{(1+A)(1-r)-1} + \frac{(1-r)^k}{(1+A)(1-r)-1} \end{aligned}$$

With reasonable network churn rate, $(1+A)(1-r)$ will be larger than 1. By ignoring the last component $\frac{(1-r)^k}{(1+A)(1-r)-1}$, which is a small constant when k is fixed, $n'(i-1)$ is bounded by $[(1+A)(1-r)]^{i-1} \times n - \mathcal{O}((1+A)^{i-k})(1-r)^i)$. Consequently, $l(i) = A \times n'(i-1) = 2^{\delta+1} \times ([(1+A)(1-r)]^{i-1} \times n - \mathcal{O}((1+A)^{i-k})(1-r)^i)$ is a sufficient condition for $l(i) \geq A \times n(i-1)$ because $n'(i-1) > n(i-1)$. Then the following analysis about the query processing cost holds based on mathematic recursion, where $i > 1$.

Without loss of generality, suppose that there are j^δ s' replicas in the network after the j_{th} period, where $1 \leq j < i$. It can be proved through mathematical induction that, after the i_{th} period, the number (denoted by $|s'|$) of s' replicas is no less than i^δ. Note that, once $|s'| \geq i^\delta$, it is easy to make $|s'| = i^\delta$ through a decentralized consensus. Specifically, during the prefetching and caching process of s', a uniform random token is generated for each such session, denoted by tok. Denote by $(0, D]$ the range domain of the uniform random function, then s' will be cached only when the token is located within $(0, \frac{D \times i^\delta}{|s'|}]$. This can guarantee that the expected number of s' replicas after the i_{th} period equals i^δ. Then the following recursive derivation holds.

$$
\begin{aligned}
l(i) \times (i-1)^\delta &\geq 2^{\delta+1} \times (i-1)^\delta \times n(i-1) && (4.20)\\
&\geq 2 \times (\frac{i}{i-1})^\delta \times (i-1)^\delta \times n(i-1) && (4.21)\\
&\geq (i^\delta + (i-k)^\delta) \times n(i-1) && (4.22)\\
\rightarrow \quad & \frac{l(i) \times (i-1)^\delta \times (1-r)}{n(i-1) \times (1-r)} - (i-k)^\delta \geq i^\delta && (4.23)
\end{aligned}
$$

Equation 4.23 shows that under the setting that the network churn rate equals r and the cached data items are evicted after k periods, the number of s' replicas (including the original and cached ones) is no less than i^δ after the i_{th} period. It is direct that, the overall cost for processing m_2 q_2 queries is no more than $\Sigma_{i=1}^{m_2} \frac{N}{i^\delta} = N + \frac{N}{2^\delta} + \frac{N}{3^\delta} + ... + \frac{N}{m_2^\delta} = \mathcal{O}(1) \times N$, even under network churn and cache expiry.

□

4.4 Performance Evaluation

4.4.1 Experimental Setting

To study range query processing performance under the prefetch-based approach, the discrete event simulator p2psim is used. A network topology is generated by randomly mapping peers to coordinates in a square area using a uniform distribution. A network of up to $N = 3000$ peers is considered. Both a static network setting and a dynamic setting with network churn are simulated.

One-dimensional range queries within the domain of $[0, 10000]$ are considered. Initially each peer holds $i = 10$ items that are randomly chosen within the range. The range query load is synthesized as follows. The number of range queries within each *epoch*[4] follows Poisson distribution with mean

[4]In this experiment, each epoch lasts 5×10^6 milliseconds.

$m = 100$; the query execution process runs for 20 epochs. To simulate the Power-law characteristics of query load that is common in real world [81], the range domain is evenly partitioned into a configurable number (*i.e.,* 10) of range segments; the start point (*i.e.,* lower bound) of each range query is generated over a randomly chosen segment based on the principles of *growth* and *preferential attachment*, which produce Power-law query distribution [11]. When a query is issued more than a number of times and becomes "popular", peers issuing it start to migrate the query (*i.e.,* issuing migrated query in the ways to be described shortly).

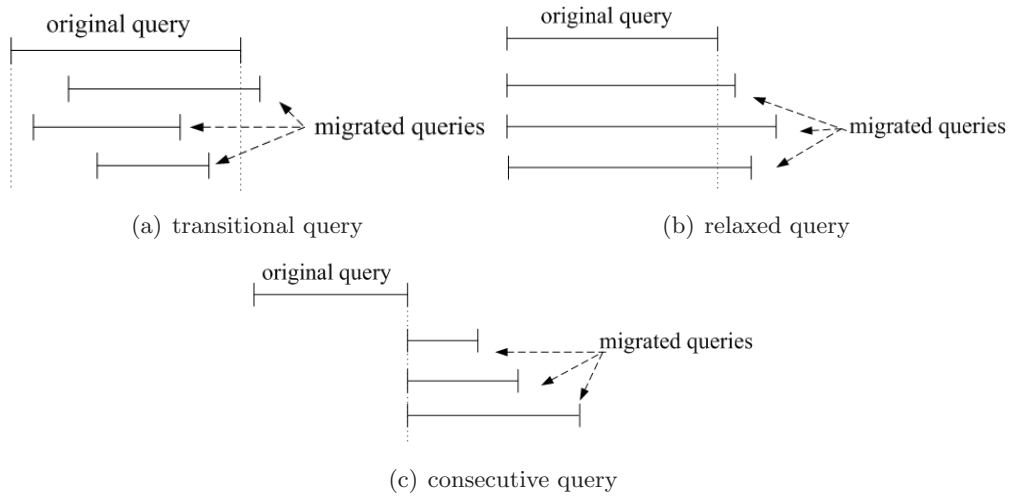
To cover various peer behaviors, three migration patterns are considered (in Figure 4.3), producing *transitional queries*, *relaxed queries* and *consecutive queries*. For each original query Q_p, a configurable number of migrated queries Q_r are generated and executed subsequently. The number (m) of the original queries and the number (m') of migrated queries per original query will be configured shortly.

- *Transitional queries*. All points in the data domain are potentially chosen as the start point of Q_r with a probability proportional to the Euclidean distance away from the start point of Q_p. The range span of Q_r follows Poisson distribution with mean $span = 100$. This migration pattern may simulate the scenario that, after finding that the hotel rooms around a specific city are all booked, peers may turn to check other proximate cities.

- *Relaxed queries*. The start point of each Q_r equals to that of Q_p, denoted by *start*. Then the range span of Q_r is relaxed to follow Power-law distribution between $(span, |D| - start)$. This migration pattern covers those relaxed queries over different radius of the region around a specific location (*e.g.,* a conference site).

- *Consecutive queries*. The start point of each Q_r adopts the end point of Q_p. The range span of Q_r follows Poisson distribution with mean $span = 100$. This migration pattern captures the setting when peers adjust the search scope by retrieving the data that are disjoint but contiguous to initial geographical range constraints.

Since the random walk mechanism requires a randomization mechanism, all experimental results are averaged over three runs, each with a different random seed. Recall that both the number of messages and the query routing latency are inversely proportional to the number of query result replicas in either flooding or random walks; therefore, only the number of messages is reported. Since only the reply message sent back to query issuers carry query results and prefetched data while all other messages only carry the query itself (*e.g.,* range constraints), the amortized message size is small. Thus, message size is not studied in the performance evaluation. In this simulation, the proportional and square-root replication schemes are considered as the baseline approaches. No overall storage constraints are assumed in this evaluation. Thus, the query processing costs incurred by the square-root replication scheme are even higher than those incurred by the proportional replication scheme, which supports the theoretic performance analysis presented in Section 4.2.



(a) transitional query (b) relaxed query

(c) consecutive query

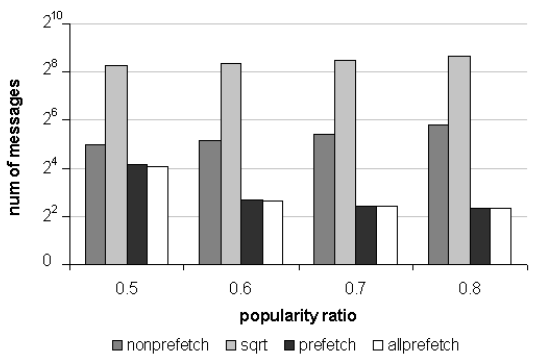
Figure 4.3: Synthesized Migrated Queries

4.4.2 Evaluation of Query Shipping

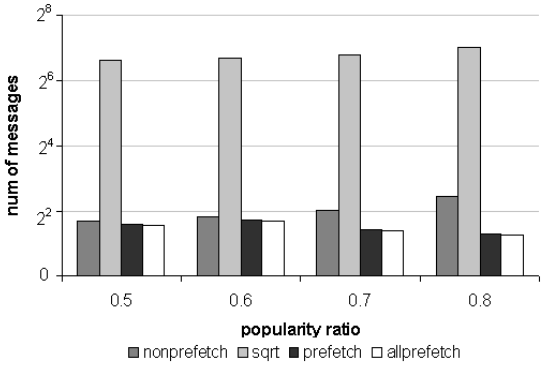
In the first experiment, a static environment is assumed, where no network churn or data insertion occur. Network churn will be considered shortly in Section 4.4.4. Initially, $m = 100$ queries are generated during each epoch by following Power-law distribution. When a query becomes "popular", $m' = 5$ migrated queries (respectively transitional, relaxed and consecutive queries) are generated based on each original query.

The average number of messages per query is shown in Figure 4.4. In addition to the *nonprefetch* option that acts the same as the proportional replication scheme, the *sqrt* option corresponding to the square root replicaiton scheme, and the *prefetch* option that corresponds to the proposed prefetch-based approach, An *allprefetch* approach is also considered, which prefetches not only correlated poorly-replicated data items but well-replicated ones during the execution of range queries. For presentation, the results are illustrated with a logarithmic scale with base of 2.

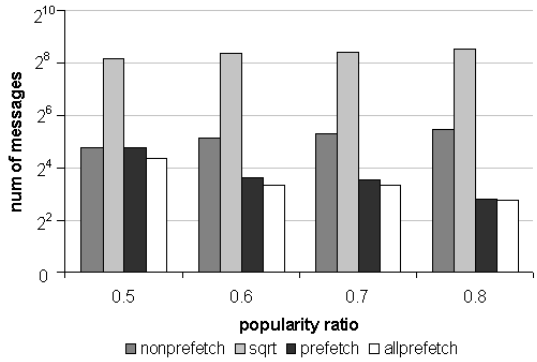
The experimental results show that the prefetch-based approach effectively decreases the overall number of messages per query with respect to all migration patterns up to 80%. In this experiment, the ratio (denoted by pr) corresponding to the x-axis of Figure 4.4 may trigger the query migration: when an original query is issued by peers for a sufficiently large number of times (*i.e.,* $c \times pr \times N$), migrated queries are generated for this query, where N denotes the network size and $c = 0.6$ for this simulation. When pr is higher, the number of original queries is expected to increase, such that more migrated queries are generated, potentially including more poorly-replicated data items; consequently the average query execution performance of the *nonprefetch* option goes up. In contrast, the average query execution cost under the *prefetch* option is lower under all settings, showing the effectiveness of the proposed prefetching approach. Figures 4.4(d)-(f) present the shipping cost consumed by migrated queries exclusively, indicating that the performance improvement of the proposed approach is primarily due to the cost savings over migrated queries. These figures also illustrate that the benefit of simply prefetching all correlated data items including



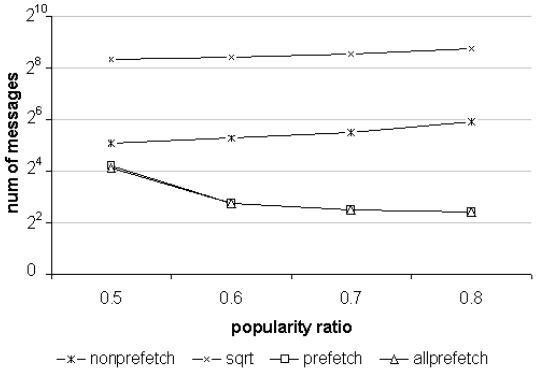
(a) transitional query overall



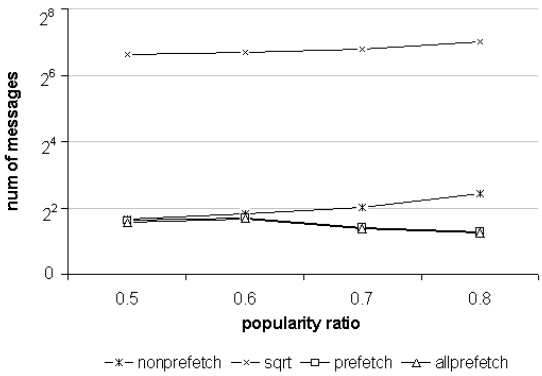
(b) relaxed query overall



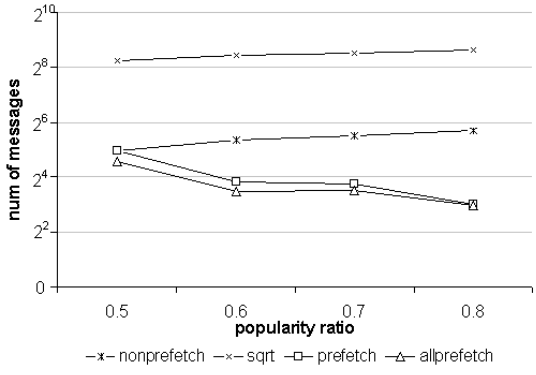
(c) consecutive query overall



(d) transitional query only



(e) relaxed query only



(f) consecutive query only

Figure 4.4: Average Shipping Cost per Query

well-replicated ones (under the *allprefetch* option) is not significant. The bandwidth consumption of the *prefetch* option is measured against that of the *allprefetch* option, which can be 60% more. This confirms the belief that data popularity should be considered in prefetching.

Since "relaxed queries" are generated by following Power-law distribution and a small set of migrated queries are issued with an exponentially higher probability, the (initially) poorly-replicated data items covered by corresponding query results become well replicated very quickly. Thus the number of messages per query is significantly lower than that of the queries generated under the other two migrated patterns. This holds for other experimental results over "relaxed queries" in the remainder of the chapter.

With respect to the load of migrated queries, A different number (*i.e.,* $m' = 1, 5$ and 10) of migrated queries are evaluated, with $pr = 0.7$. As shown in Table 4.1, the average number of messages decreases because a relatively larger number of queries (over poorly-replicated data items) may benefit from the prefetching of poorly-replicated data.

Migrated queries	$m' = 1$	5	10
transitional	10.47	5.39	4.61
relaxed	2.66	2.31	2.30
consecutive	9.93	7.59	7.59

Table 4.1: Query Execution Cost with Different Query Load

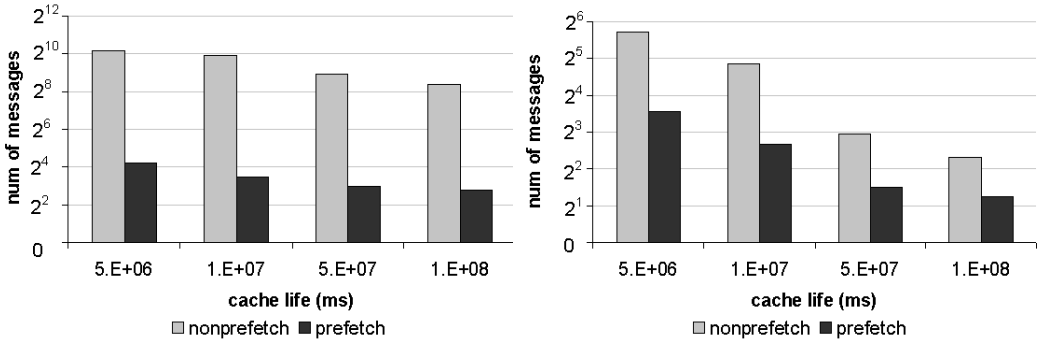
4.4.3 Evaluation with Cache Constraints

In practical systems, peers usually have caches with limited storage sizes. Moreover, cached data items may expire after a period of time, as discussed in Section 4.3. In this simulation, the query shipping cost with respect to different cache constraints is measured.

Cache sizes (*i.e.,* the cached data items per peer) of 500, 1000, 1500 and 2000 are considered. The experimental results are shown in Table 4.2, which indicate that when the cache size is larger, the average query shipping cost tends to be lower. This is primarily due to the fact that larger caches can hold more prefetched data, facilitating query execution. Query processing performance is also evaluated under various cache expiry periods (*i.e.,* 5×10^6, 1×10^7, 5×10^7 and 1×10^8 simulation milliseconds). The results (shown in Figure 4.5) demonstrate that when the lifespan of cached data items increases, the average number of messages per query decreases. This is because longer cache life means that data expire less frequently, leading to better use of cached data items.

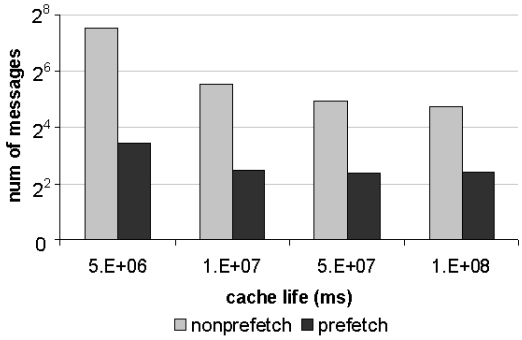
Migrated queries	$s = 500$	1000	1500	2000
transitional	357	321	255	241
relaxed	2.73	2.11	2.07	2.07
consecutive	72	24	16.8	12.19

Table 4.2: Query Execution Cost under Various Cache Sizes



(a) transitional query

(b) relaxed query



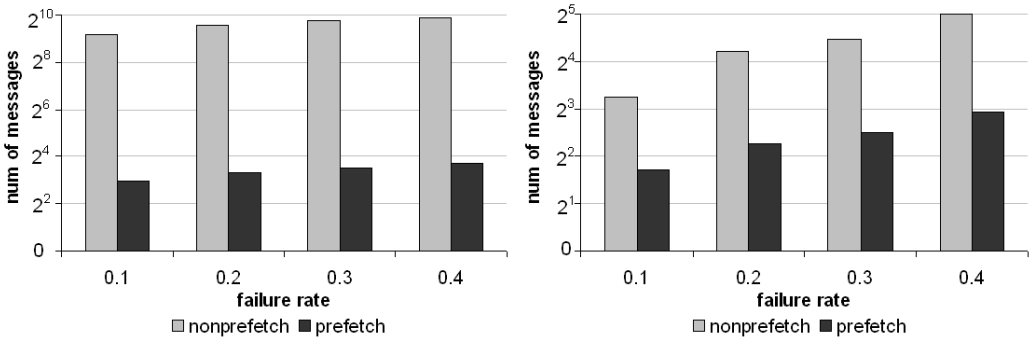
(c) consecutive query

Figure 4.5: Query Execution Cost per Query under Cache Expiry

4.4.4 Evaluation under Dynamic Settings

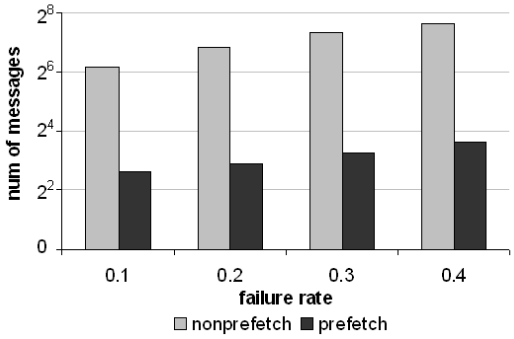
In P2P networks, peers may leave and fail arbitrarily, affecting the number of cached data items in the network. Moreover, peers may join the network with new data items and data insertion manipulation may also be issued by existing peers.

Network churn is simulated by allowing each peer to leave with a probability of r per epoch, where r varies from 0.1 to 0.4. To make the network stable, peers join the network at the same rate. The number of messages per query is shown in Figure 4.6. The experimental results indicate that when failure rate increases, the number of messages per query goes up, which is due to more random walks being required for completing the query processing during network churn.



(a) transitional query

(b) relaxed query



(c) consecutive query

Figure 4.6: Query Execution Cost per Query with Peer Failure

To measure range query processing performance when data insertion occurs, peers generate a configurable number (*i.e.,* 1, 10, 100 and 1000 per epoch) of new data items that are randomly chosen within the data domain. In this experiment, no migrated queries are generated. As shown in Figure 4.7, the average shipping cost per query increases steadily during the insertion of new data items. This is not surprising, because when peers keep issuing the same range queries, they still need to locate the new data items that initially are poorly-replicated. Similar to data insertion, the deletion of data items may also incur poorly-replicated data items, affecting range query execution performance in a similar way. Thus, data deletion will not be studied separately. Moreover, since

m'	1	10
transitional (*byte*)	204,829	183,496
relaxed (*byte*)	3274	1402
consecutive (*byte*)	72,836	62,960

Table 4.3: Bandwidth Consumption per Query with Different Migration Queries

data items are assumed to be read-only (*e.g.,* song files or hotel address information), data insertion or deletion does not incur inconsistency among cached data replicas. Thus data consistency is not an issue in this context.

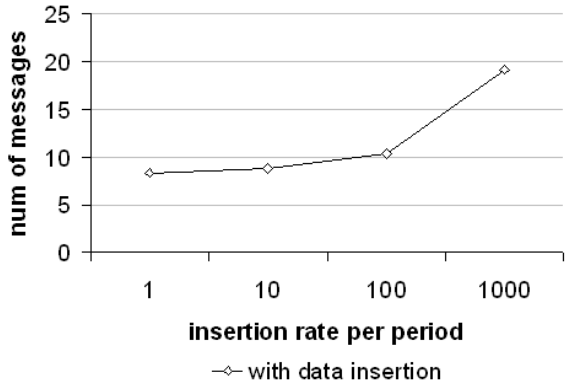


Figure 4.7: The Number of Messages per Query with Data Insertion

4.4.5 Evaluation of Bandwidth Overhead and Adaptivity

The correlation threshold τ is computed adaptively based on the knowledge of executed range queries. In this performance studies, the bandwidth costs of prefetching with respect to a query load consisting of the same number of queries but with different migration query loads: either $m' = 1$ migrated query is generated for a "popular" original query or $m' = 10$ migrated queries are synthesized. The results shown in Table 4.3 indicate that the bandwidth consumed by the prefetch-based approach with a relatively larger number of migrated queries is higher. This is because the execution of more queries is affected by poorly-replicated data items and τ tends to be larger, increasing the volume of prefetched data. In contrast, when migrated queries are fewer, the value of τ is smaller. Thus the volume of perfetched data is lower, demonstrating the adaptivity of the proposed approach.

4.5 Summary

Under unstructured P2P overlay network architecture, query processing performance is usually decided by the number of query result replicas in the network. When range queries involve poorly-replicated data items, query execution performance degrades. In this chapter, a popularity-aware prefetch-based caching approach is proposed that effectively facilitates the caching of poorly-replicated data items that are correlated with well-replicated ones, resulting in cost savings for future queries that access the poorly-replicated data. The approach does not require strong altruistic cooperation from peers since only query issuers that use the query processing services incur prefetching overhead. It is proven that, under a certain query load setting, the performance of query processing that involves poorly-replicated data items can be quantitatively improved. Extensive simulation also demonstrates that the prefetch-based approach decreases the query processing cost substantially under various query load settings.

Chapter 5

Gossip-based Approach for Join Query Processing in Unstructured P2P Networks

In modern massively distributed networks, data normally originate from multiple sources and their integration for large-scale sharing is an important issue. For instance, emerging mashup Web services such as Chicago Crime Map[1] and NaviTraveller[2] join their Web service data with the geographic data from GoogleMaps[3] to support enhanced services.

Under unstructured P2P architecture, data integration over heterogeneous data can be modeled as join query processing, which potentially supports applications such as content distribution and distributed location-aware services.

A novel two-phase join query operator is proposed in this chapter that efficiently propagates data distribution information with respect to join attributes, which enables peers to self-organize into groups such that join query processing can be conducted independently within each group. In this chapter, based on random-sampling techniques, efficient mechanisms are developed to discover group members. These mechanisms are guaranteed to complete in a bounded and scalable number of communication rounds. It is shown that the proposed approach is both effective and efficient for distributed join query processing in P2P networks. Especially, under a setting where data that are close in data space are clustered on peers such that join query processing involves only a subset of peers, the proposed approach tends to be much more economical than existing approaches regarding bandwidth consumption and query processing latency.

Note that, when data that are close in data space are clustered on peers, order-preserving structured overlay networks (*e.g.,* P-Grid) can be employed as a routing substrate to support join query processing. However, this chapter focuses more on unstructured P2P networks without explicitly building structured overlays or migrating data among peers.

[1] http://www.chicagocrime.org

[2] http://www.navitraveler.com/

[3] http://maps.google.com

5.1 Overview of Design Principles

Since heterogeneous data may be distributed across multiple peers, the join query operator is essential for complex query processing in various applications. Consider a P2P publication/subscription (pub/sub) system [1, 3] over distributed video data. Each peer may act as a publisher that shares home-made video or as a subscriber that intends to browse the video data of others. Suppose that subscribers declare their preferences via attribute *"pref"*, and publishers mark up their video data with semantic categorizations, denoted by attribute *"cat"*. Once the semantic categorization of certain data hosted on a publisher peer matches the preference of a subscriber peer, the latter will maintain the physical address of the former for subsequent data fetching purpose. P2P networks are usually dynamic and peers may join and leave the network arbitrarily: leaving publishers may become unavailable such that the data published at these peers are lost; conversely, peers joining the network may become publishers that share new data. In addition to network churn, publishers may change their semantic categorizations while subscribers may also update their preferences. All these behaviors may invalidate the current subscriptions. Thus peers should periodically update their subscriptions to improve user experiences and system performance. Conceptually, denote by S the logical relation that covers all the tuples on subscriber information while denote by P the logical relation that unions the tuples of publisher information. Then the subscription updating process is equivalent to the running of the following join query, where *ipAddr* denotes the physical address of the publisher peers. Note that, in this example, the join query operation is enforced only over the metadata rather than actual video data.

SELECT S.ipAddr, P.ipAddr, S.pref

FROM S, P

WHERE S.pref = P.cat

Two major issues exist in implementing the join query operator in P2P networks: decentralized catalog management and efficient query processing. First, P2P networks are usually large-scale and dynamic such that employing centralized catalogs to manage the data placement information of all peers would not scale. Since data on arbitrary peers may join to produce results, without catalogs, it is unclear how peers can efficiently discover relevant data for query processing. Second, in massively distributed networks such as the Internet, controlling communication cost is crucial for the viability of the system since the communication cost affects overall performance of in-network query processing. In addition to communication cost, local processing cost may also affect query processing latency when it becomes non-trivial.

A few approaches exist for distributed join query processing over P2P networks. However, they either make strong assumptions on the availability of catalog information [13, 75], which is hard to achieve in large-scale dynamic P2P networks, or they rely on specific routing infrastructures (*e.g.,* DHT), which requires numerous messages to achieve DHT-based routing [47].

In this chapter, a purely decentralized two-phase join query operator is proposed that considers both decentralized catalog management and efficient query processing. During the first phase, via efficient decentralized propagation mechanisms, peers simultaneously propagate compact meta-

information about the distribution of their data over the join attributes (referred to as "data distribution information" in the remainder of this chapter), such that they can self-organize into groups, each corresponding to a non-overlapping partition of the data space over the join attributes. During the second phase, all groups run bandwidth-efficient join query processing concurrently and ship query results to the query issuer where all results are aggregated to complete the query processing.

This approach is advantageous in purely decentralized unstructured P2P networks because: (1) it does not depend on centralized or fully-replicated catalogs; instead, an efficient group discovery mechanism is developed such that peers self-organize into groups for parallel join query processing; (2) based on the collected data distribution information, unnecessary query processing is avoided (*e.g.,* for those groups that lack join operands), which is not achieved by existing approaches; (3) the group-wise join query processing exploits intra-operator parallelism, which is feasible in P2P networks where peers have sufficient computational capacities to conduct query processing concurrently; and (4) depending on application requirements, the proposed approach can adaptively choose various data propagation and join query processing techniques; in this work the efficient and fault-tolerant uniform gossip protocol [31] is used for data propagation and the bandwidth-efficient Bloom join [62] is used for data integration within groups. Finally, the proposed approach is guaranteed to obtain complete join query results. Thus the corresponding join query operator can be easily applied in a general P2P query processing framework.

5.2 Partition-based Join Query Processing

Initially, peers obtain join queries so that they can consistently extract join attributes and construct a data space with each dimension corresponding to a join attribute. In addition to obtaining the join attribute information explicitly from join query statements, such information can also be derived implicitly via foreign key constraints. Join attribute information can be populated to each peer during the time it joins the overlay network. Alternatively, such information can be propagated to all peers through multicast, as employed in the PIER system [47].

Since overall data distribution is hard to obtain in large-scale and dynamic P2P networks, space partitioning strategy is employed. Each peer evenly divides each dimension of the data space into s partitions, where s is a configurable system parameter (usually a small constant for scalability). Suppose that there are d join attributes, then the total number of partitions is s^d. Note that, when two partner tables join over an attribute, their attribute names may not be exactly the same. For example, in the P2P content pub/sub application discussed in Section 5.1, join query operation is imposed over different attribute names $S.pref$ and $P.cat$. However, it is straightforward to map them to an identical join attribute name (*e.g.,* a concatenated attribute name $pref\_cat$), so that the definition of the data space over all dimensions is uniform across all peers, regardless of which partner tables they are hosting.

The actual data distribution among peers may be misaligned with the data space partitions such that each peer may correspond to multiple partitions. As shown in Figure 5.1(a), the shadowed

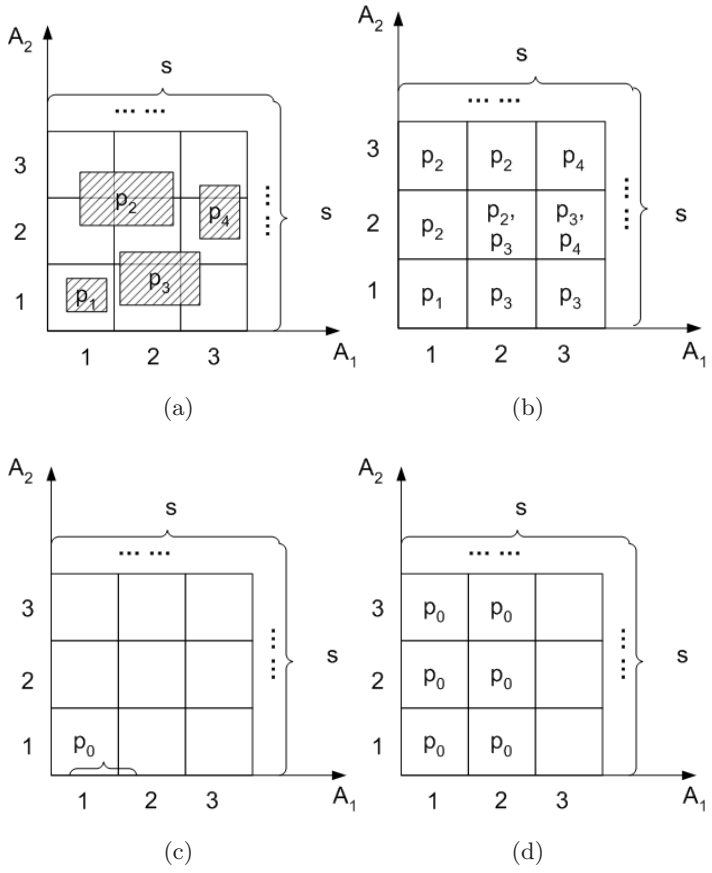


Figure 5.1: Data Space Partition

blocks dotted in the evenly partitioned two-dimensional space (over join attributes A_1 and A_2) denote the minimum bounding boxes of the data held by the corresponding peers. Obviously, the data on the peers such as $\mathcal{P}_2$, $\mathcal{P}_3$ and $\mathcal{P}_4$ cross multiple partitions, as shown in Figure 5.1(b).

For ease of presentation, in the remainder, all peers that correspond to the same partition constitute a *group*. Since data on some peers may not be defined over all dimensions, more rigorously, *a peer belongs to a group if its data intersect with the corresponding partition ranges on all the dimensions where the data are defined*. For example, as shown in Figure 5.1(c), peer $\mathcal{P}_0$ hosts the data items that are defined only over the join attribute A_1 within the first two partitions (*i.e.,* $A_1 = 1$ and $A_1 = 2$). Thus all groups covering the two A_1 partitions will include $\mathcal{P}_0$ (see Figure 5.1(d)). This is crucial for join query processing since $\mathcal{P}_0$'s data may join with any data whose projection results over A_1 are located within the first two partitions, regardless of the values over the other attributes such as A_2.

Given a query, peers start the grouping process by simultaneously propagating their data distribution information over join attributes. Various propagation mechanisms exist; however, many incur a communication cost that is polynomial to network size N. For example, a naive flooding issued from each peer takes $\mathcal{O}(N^2)$ messages and $\mathcal{O}(N)$ communication rounds, where a peer exchanges at most one message during each communication round. For better communication efficiency, uniform gossip mechanism [31] is employed, which takes $\mathcal{O}(\ln N)$ gossip rounds and $\mathcal{O}(N \ln N)$ messages for N peers to propagate messages among themselves with high probability. To realize gossip-based propagation, certain unspecified synchronization mechanisms are assumed to exist, such as an approximate global clock, which can be based on network-based time protocols [67], or even GPS. If such synchronization mechanisms do not exist, faster peers can easily re-synchronize by making all requests (*e.g.,* to start gossiping) to be blocking requests.

Data distribution information can be customized according to specific applications. In this dissertation, data distribution information consists of a peer's IP address and the unique identifiers of the groups that it belongs to. Although the propagation of such information is guaranteed to complete in $\mathcal{O}(\ln N)$ rounds through gossiping, due to the accumulation of IP addresses, the size of each message may eventually become proportional to N. This makes the approach unscalable to large-scale networks. Thus, two efficient mechanisms are proposed that can discover group members in a purely decentralized manner *with message size independent of the network size*. The proposed mechanisms only require that each peer maintains a scalable set of random chosen peers over the whole unstructured P2P network, which can be obtained efficiently through well-established sampling approaches [41, 54].

After each peer learns the members of its corresponding groups, group propagation can be conducted efficiently. Bandwidth-efficient techniques such as Bloom join [9, 92] are employed over the data within each group. Moreover, since each group produces join query results independently, the proposed approach may facilitate query processing by exploiting intra-operator parallelism (see Section 5.2.3).

5.2.1 Discover Group Cardinality

Random sampling services are realized in unstructured P2P networks [41, 53]. Equivalently, each peer can maintain a scalable number of random peer addresses all over the network. With these random samples, uniform gossip protocol can be realized.

Any peer $\mathcal{P}$ learns its affiliated groups and the corresponding partitions by intersecting its data with the partitioned data space. All peers then simultaneously initiate *push-sum protocol* [56] to compute *group cardinalities* (*i.e.,* the number of peers within each group). To compute the cardinality of the data that are distributed on peers, the push-sum protocol requires each peer to simultaneously propagate a weight value and the weighted sum of all obtained cardinality information over the whole network for multiple rounds. With a probability of at least $(1-\epsilon)$, all peers obtain the data cardinality with an approximation error bounded by δ after $\mathcal{O}(logN + log\frac{1}{\epsilon} + log\frac{1}{\delta})$ gossip rounds, where both δ and ϵ can be configured to be very small [56].

At the start of the group cardinality computation process, each peer $\mathcal{P}$ sets to 1 the initial cardinality values of those groups that it belongs to, and the cardinality values of the other groups to 0. Then all peers run the push-sum protocol concurrently. The computation process for any peer $\mathcal{P}$ is presented in Algorithm 7, where gc_i^x denotes the group cardinality value for group G_i at the x_{th} gossip round, and w_i^x denotes the corresponding weight value. Since cardinality information over all groups is packaged in each message, the space overhead of each message is proportional to the number of groups (*i.e.,* at most s^d), independent of the network size. After the computation, each peer only maintains the cardinalities of those groups that it belongs to. In many applications, such as location-aware services, peers usually manage data that are geographically clustered. This results in a limited number of groups, reducing the storage cost to maintain the group information. The propagation process is scalable with respect to the network size N, taking $\mathcal{O}(logN + log\frac{1}{\epsilon} + log\frac{1}{\delta})$ gossip rounds, with bandwidth cost proportional to $\mathcal{O}((logN + log\frac{1}{\epsilon} + log\frac{1}{\delta}) \cdot s^d)$.

5.2.2 Discover Group Membership Information

In addition to group cardinalities, each peer also needs to learn group membership information for communication within groups. For scalability, it is sufficient for each peer to maintain $\mathcal{O}(\ln|G|)$ random samples of each group G that it belongs to, which enables gossip-based group communication, where $|G|$ denotes the group cardinality of G.

Remember that each peer already learns the information of random samples over the whole network. A novel scalable polling-based strategy is developed for peers to learn the random samples of groups. Specifically, each peer $\mathcal{P} \in G$ first multicasts a membership-check message to its random samples (called *contact*) over the whole network. Subsequently, each contact concurrently and recursively polls its own contacts. Membership-check messages only carry group identifier information, which is compact and independent of network size. Moreover, the polling process does not involve message exchanges such that it can complete in one communication round, called "polling round". If any contact belongs to G, it will reply to $\mathcal{P}$ with its IP address and be recruited as $\mathcal{P}$'s sample regarding G. The polling process for $\mathcal{P}$ to learn $\ln|G|$ random samples of group G

Algorithm 7 *compute_cardinality*()

1: //initialization; round 0
2: **for** each group G_i **do**
3: **if** $\mathcal{P}$ belongs to G_i **then**
4: $gc_i^0 \leftarrow 1$;
5: **else**
6: $gc_i^0 \leftarrow 0$;
7: **end if**
8: $w_i^0 \leftarrow 0$ *or* 1; //one peer gets w^0 as 1, others as 0
9: **end for**
10: //iteration until the procedure converges
11: $r \leftarrow 1$;
12: **for** each group G_i **do**
13: //push-sum protocol
14: $gc_i^r \leftarrow \frac{1}{2} \cdot \Sigma gc_i^{r-1}$;
15: $w_i^r \leftarrow \frac{1}{2} \cdot \Sigma w_i^{r-1}$;
16: send gc_i^r and w_i^r to a randomly chosen peer;
17: //after $\mathcal{O}(logN + log\frac{1}{\epsilon} + log\frac{1}{\delta})$ rounds
18: **if** gc_i^r and w_i^r converge **then**
19: group cardinality for group G_i is equal to $\frac{gc_i^r}{w_i^r}$;
20: **end if**
21: $r \leftarrow r + 1$;
22: **end for**

is described in Algorithm 8.

Algorithm 8 *sample_group_with_polling*(G)

```
1: m ← ln|G|;
2: for each of the O(ln N) rounds do
3:    P randomly polls its direct or recursive contact P';
4:    if P' belongs to G && samples are fewer than m then
5:       P keeps P' as a random sample of G;
6:    end if
7: end for
```

Generally, suppose that peer $\mathcal{P}$ needs to learn m random samples of a group G, where m is a configurable parameter. Define a random variable X_i as the event that the i_{th} sample peer belonging to G is obtained by uniform random sampling over the whole network (called *global sampling* below). The probability to choose the first sample of group G by a global sampling is $\frac{|G|}{N}$, which is independent of other global samplings. Iteratively, the probability to choose the i_{th} distinct sample of group G by a global sampling equals to $\frac{|G|-i+1}{N}$. Thus the expected number of X_i is $E[X_i] = \frac{N}{|G|-i+1}$ and an upper-bound of the expected number of global samplings to obtain m samples is proved in Lemma 9.

Lemma 9. *The expected number of global samplings to obtain m samples is bounded by $N \cdot \mathcal{O}(\ln m)$.*

Proof.

$$\begin{aligned} E[X] &= E[\Sigma_{i=1}^{m} X_i] = \Sigma_{i=1}^{m} E[X_i] \\ &= N \cdot (\frac{1}{|G|} + \frac{1}{|G|-1} + ... + \frac{1}{|G|-m+1}) \\ &\leq N \cdot (\frac{1}{m} + \frac{1}{m-1} + ... + 1) \\ &\approx N \cdot \mathcal{O}(\ln m) \end{aligned}$$

□

When $m = \mathcal{O}(\ln |G|)$, which is sufficient to enable the uniform gossip protocol within the group, $E[X] \leq N \cdot \mathcal{O}(\ln \ln |G|)$ is upper-bounded by $N \cdot \mathcal{O}(\ln \ln N)$. Since each peer already learns $\mathcal{O}(\ln N)$ contacts over the whole network, and each polling is equivalent to a global sampling, it takes $\mathcal{O}(\ln N)$ concurrent polling rounds to fulfill $N \cdot \mathcal{O}(\ln \ln N)$ global samplings for each peer to obtain the random samples of group G, as proved below.

Theorem 10. *It takes at most $\mathcal{O}(\ln N)$ concurrent polling rounds for each peer to obtain $\mathcal{O}(\ln |G|)$ random samples for the group G it belongs to.*

Proof. Define the number of rounds to be x. Suppose that each peer knows k random samples over

the whole network.

$$\Sigma_{i=1}^{x} k^x = k + k^2 + ... + k^x = \frac{k \cdot (1 - k^x)}{1 - k} = N \cdot \ln \ln N \tag{5.1}$$

$$\rightarrow k^x = \frac{k-1}{k} N \ln \ln N + 1 \tag{5.2}$$

$$\rightarrow k^x \leq N \ln \ln N \tag{5.3}$$

$$\rightarrow x \leq log_k(N \ln \ln N) = \frac{\ln N}{\ln k} + \frac{\ln(\ln \ln N)}{\ln k} \tag{5.4}$$

Because the second component of the equation 5.4 can be regarded as a small constant, $x \leq \mathcal{O}(\ln N)$. Consequently, $x = \mathcal{O}(\ln N)$ polling rounds will produce $\mathcal{O}(N \ln \ln N)$ global samplings, after which $\mathcal{P}$ is expected to get $\mathcal{O}(\ln |G|)$ random samples of group G. □

Denote by c the maximum number of groups that a peer can belong to. All peers initiate the polling process concurrently. After at most $c \cdot \mathcal{O}(\ln N)$ polling rounds, each peer obtains random samples for each affiliated group.

Since all N peers simultaneously conduct the group discovery process, the overall number of global-sampling messages becomes quadratic to the network size (*i.e.,* $\mathcal{O}(N^2 \ln \ln N)$). Compared with other propagation mechanisms that require an equivalent number of messages, such as naive flooding, the proposed approach consumes far fewer communication rounds, and the messages are compact, only containing group identifiers. However, due to bandwidth constraints, some applications may prefer fewer messages. An alternative approach is now addressed in the following, which takes at most $N \cdot \mathcal{O}(\ln |G_{max}|) \cdot s^d$ messages with an equivalent message size and approximately $c \cdot \mathcal{O}(\ln N)$ polling rounds.

Each group randomly chooses a *representative* through a *decentralized consensus* process. The representative conducts a polling process to learn full group membership information and propagates it to all group members. Specifically, each peer belonging to an arbitrary group G generates a random token through a globally-consistent hash function such as SHA-1 [89], and one peer whose token is located between $(0, \frac{D}{|G|}]$ is chosen as the representative of group G, denoted by $\mathcal{P}_G^{rep}$, where D is the range of the hash function. Each $\mathcal{P}_G^{rep}$ then initiates the polling process simultaneously and obtains full membership information (*i.e.,* $m = |G|$ "samples") for the group. Based on the same derivation as shown in the proof of Theorem 10, $\mathcal{P}_G^{rep}$ is able to collect the full membership information of G in $c \cdot \mathcal{O}(\ln N)$ gossip rounds.

Theorem 11. *After $c \cdot \mathcal{O}(\ln N)$ gossip rounds, all representatives collect the full membership information of the groups that they belong to, taking overall $N \cdot \mathcal{O}(\ln |G_{max}|) \cdot s^d$ membership-check messages.*

Proof. According to Lemma 9, the expected number of global samplings to obtain full membership information (*i.e.,* at most $|G_{max}|$ "samples") is bounded by $N \cdot \mathcal{O}(\ln |G_{max}|)$. Suppose each peer knows $k = \ln N$ contacts and denote by x the number of rounds for each representative to obtain

the full membership information of the group that it belongs to.

$$\Sigma_{i=1}^{x} k^x = k + k^2 + ... + k^x = \frac{k \cdot (1 - k^x)}{1 - k} = N \cdot \ln |G_{max}|$$
$$\rightarrow k^x = \frac{k-1}{k} N \cdot \ln |G_{max}| + 1$$
$$\rightarrow k^x \leq N \cdot \ln |G_{max}|$$
$$\rightarrow x \leq log_k(N \cdot \ln |G_{max}|) = \frac{\ln N}{\ln \ln N} + \frac{\ln(\ln |G_{max}|)}{\ln \ln N}$$

Thus, after $\mathcal{O}(\ln N)$ polling rounds, each representative obtains the full membership of a group that it belongs to. Since each representative is affiliated with at most c groups, it holds that, after at most $c \cdot \mathcal{O}(\ln N)$ polling rounds, the membership information of all groups can be obtained by all representatives.

Since the process to obtain the full membership of each group consumes $N \cdot \ln |G_{max}|$ membership-check messages, the overall number of messages to complete the process for all s^d groups is bounded by $N \cdot \mathcal{O}(\ln |G_{max}|) \cdot s^d$.

□

After this, $\mathcal{P}_G^{rep}$ simply multicasts partial or full membership information to all other group members. In contrast to the previous one, this approach trades off the communication cost against both load-balancing (because representatives take all the workload) and fault-tolerance (because representatives are subject to single point of failure). Since the multicast with each group consumes at most $|G_{max}|$ messages, the overhead number of messages to propagate the group membership information to all peers is bounded by $|G_{max}| \cdot s^d$.

For clarity, the process of this alternative sampling approach is presented in Algorithm 9.

Algorithm 9 *sample_group_with_polling_alt*(G)

```
1: each representative P ∈ G issues the sampling process independently;
2: for each of the O(ln N) rounds do
3:    P randomly polls its direct or recursive contact P';
4:    if P' belongs to G && samples are fewer than |G| then
5:       P keeps P' as a random sample of G;
6:    end if
7: end for
8: each representative propagates the membership information to all peers within the group;
```

Generally, the group discovery mechanism (including the sampling mechanisms) can be applied in ad-hoc fashion such that the groups are dismissed immediately after the query processing. Otherwise, the group member information can be cached by peers for reuse for similar queries until it expires as regulated.

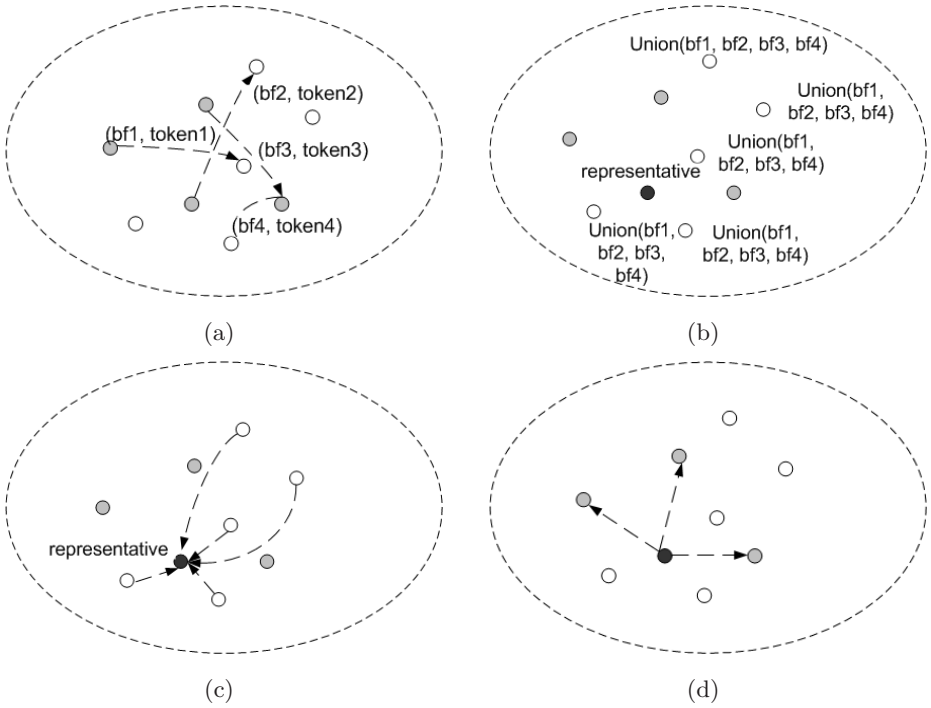


Figure 5.2: Group-wise Process of a Join Operator

5.2.3 Bloom Join Query Processing

After the group discovery phase, each peer learns at least $\mathcal{O}(\ln |G|)$ random samples of each affiliated group G, which enables gossip-based propagation mechanisms within G.

Since join query processing over massively distributed networks may involve large volumes of data, bandwidth efficiency of the approach is a concern. Thus Bloom join techniques are considered in group-wise join query processing. However, other techniques such as nested-loop join and sort-merge join query processing mechanisms are applicable as well, depending on application requirements.

An advantage of using Bloom-filters with the gossip protocol is that they are duplicate-insensitive. During the gossip process, when receiving Bloom-filters, each peer $\mathcal{P}$ simply super-imposes all Bloom-filters with a Bit-Or operation. Regardless of duplicate Bloom-filters, this super-imposition does not introduce false negatives, guaranteeing the correctness of the produced join query results. The bit-length of the Bloom-filters is configured by each peer when it joins the overlay network, which decides the false positive ratio of the Bloom-filters[4]. More adaptively, the cardinality computation approach discussed in Section 5.2 can be used in the mean time to estimate the number of tuples of each join operand with respect to each group, such that peers can communicate and choose a bit-length according to a desired false positive ratio.

Regarding join queries, different join attributes may have different join selectivities and the

[4]The false positive probability of Bloom-filter approximates $0.6185^{\frac{x}{y}}$, where x is the bit-length, and y is the number of data items to be encoded [16].

ordering of join attributes may affect query processing performance. For simplicity, at this moment it is assumed that all groups have an ordering of the multiple join attributes. Selectivity-based join attribute ordering will be discussed in Section 5.3.

Consider a group G and a join attribute A. Based on an existing synchronization mechanism, all peers that carry the tuples of one join operand (*e.g.,* the smaller partner table) produce Bloom-filters simultaneously and propagate them via the gossip protocol within the group. After $\mathcal{O}(\ln |G|)$ rounds, all group member peers are expected to obtain the super-imposed Bloom-filter over all the Bloom-filters that are propagated within the group. Each peer that carries the tuples of the bigger partner table then matches the super-imposed Bloom-filter against its local tuples; those tuples that are negative to the Bloom-filter over A are removed from subsequent join query processing. Before the propagation of the Bloom-filters, each peer also generates a random token and piggybacks this token together with its IP address during the Bloom-filter propagation process. Thus at the end of the propagation, all group members also obtain a consistent set of tokens. The peer corresponding to the smallest token acts as a *representative*. All peers then that carry the tuples from the bigger partner table deliver the remaining tuples that passed the Bloom-filtering test to the representative. In a subsequent round, all peers that host data belonging to a smaller table fetch tuples from the representative and complete the join query processing over join attribute A, generating intermediate tuple results that will participate in the join query operation over the next join attribute if available.

For clarity, join query processing is illustrated in Figure 5.2. Consider a group of peers over join attribute A. From conceptual level, one partner table of a join operator aggregates all the tuples in the network that belong to the corresponding relation. Then denote by grey nodes the member peers that host the tuples from a smaller partner table, and denote by white nodes the peers hosting the tuples from a bigger partner table. In Figure 5.2(a), all grey nodes generate Bloom filters over the projection values of the local tuples over attribute A (denoted by bf_i) and produce tokens $token_i$. Both the Bloom-filter and token information is propagated via gossip protocol within the group. In Figure 5.2(b), all the white nodes receive the Bloom filters and tokens, and they independently union the Bloom filters and decide the representative (marked as a black node) through decentralized consensus process (described in Section 5.2.2). At this step, all the grey nodes also receive the tokens, so that they will choose the same representative consistently. Subsequently, in Figure 5.2(c), all the white nodes ship the tuples that passed the Bloom-filtering test to the representative node; the tuples are then fetched by grey nodes, as shown in Figure 5.2(d). Finally, all grey nodes generate the intermediate tuple results that will constitute an intermediate relation to be treated as one of the partner tables of the next join operator if available.

This approach is economical in terms of bandwidth consumption because only those data necessary for join query processing are shipped over the network. The employment of representatives does not degrade the fault-tolerance of the system since they are randomly chosen. However, these representatives may become overloaded when their interactions (*i.e.,* uploading and downloading of tuples) with group members are intensive. In such a case, multiple representatives can be voted during the decentralized consensus process, with each representative in charge of a disjoint sub-domain over attribute A within G's partition, alleviating the load-balancing problem.

For clarity, the group-wise join query processing of each peer $\mathcal{P}$ over the join attribute A is described in Algorithm 10. To recognize which partner table is smaller, the cardinalities of tables affiliated with specific join attributes are piggybacked with group cardinality information during the group discovery process that has been addressed in Section 5.2 (*i.e.,* Algorithm 7).

Algorithm 10 *Bloom_join_query_processing*()

1: **if** $\mathcal{P}$'s data belong to a smaller partner table **then**
2: generate a Bloom-filter over A and propagate it;
3: **end if**
4: Gossip the Bloom-filters among all group members;
5: **if** gossip propagation completes && $\mathcal{P}$'s data belong to the bigger parter table **then**
6: match the Bloom-filter against the data over A;
7: send the intermediate results to the representative peer;
8: **end if**
9: **if** $\mathcal{P}$ is the representative peer **then**
10: collect intermediate results;
11: wait for peers (carrying the tuples from the smaller partner table) to fetch the tuples and join with their data;
12: **end if**

Bloom join query operation is executed over each join attribute. Once the operations complete over all join attributes, the produced results within all groups are shipped to query issuer, where the results are aggregated via union operation to generate final results. When each group produces correct (*i.e.,* complete and sound) join query results, it is proven that the overall aggregated results are also correct. Recall that, the completeness of query results is defined with respect to "static snapshot" [10], during which network churn and data updates that affect the query results are ignored.

Theorem 12. *The union of the correct (i.e., complete and sound) query results obtained over all groups produces correct final results.*

Proof. Without loss of generality, consider an l-way join query $R_1 \bowtie_{A_1} R_2 \bowtie_{A_2} ... \bowtie_{A_{l-1}} R_l$ with join attributes $A_1, A_2, ...,$ and A_{l-1}. Suppose that the domain of each join attribute A_i is partitioned into s_i disjoint but contiguous partitions. Then an arbitrary relation R_i is equivalent to the union of disjoint sub-relations defined over its join attribute A_j, denoted by $R_i = R_i^{A_j^1} \cup R_i^{A_j^2} \cup ... \cup R_i^{A_j^{s_j}}$, where $1 \leq i \leq l$ and A_j^x represents the x_{th} partition over A_j, $1 \leq x \leq s_j$. Based on the *distributive property* of join query operator, $R_1 \bowtie_{A_1} R_2 \bowtie_{A_2} ... \bowtie_{A_{l-1}} R_l = \cup_{i_1 \in [1,s_1],...,i_l \in [1,s_l]} (R_1^{A_1^{i_1}} \bowtie_{A_1} R_2^{A_1^{i_1}, A_2^{i_2}} \bowtie_{A_2} ... \bowtie_{A_{l-1}} R_l^{A_{l-1}^{i_l}})$, where $R_j^{A_j^{i_j}, A_{j+1}^{i_{j+1}}}$ represents the partition of relation R_j over both the consecutive join attributes $A_j^{i_j}$ and $A_{j+1}^{i_{j+1}}$; each item $R_1^{A_1^{i_1}} \bowtie_{A_1} R_2^{A_1^{i_1}, A_2^{i_2}} \bowtie_{A_2} ... \bowtie_{A_{l-1}} R_l^{A_{l-1}^{i_l}}$ is bijective (*i.e.,* one-to-one correspondent) to a disjoint partition of the data space, while the union of all items covers the whole data space. Thus when the query results obtained over each partition are correct

(*i.e.,* complete and sound), the aggregated results over all partitions are also guaranteed to be correct. □

Recall that $|G_{max}|$ denotes the maximum cardinality among all groups. The Bloom join query processing takes $\mathcal{O}(\ln |G_{max}|)$ gossip rounds to complete, which however disregards that some peers may belong to multiple groups. Instead of taking part in the join query processing of one group till its completion, each peer participates the query processing within all its affiliated groups in a round-robin fashion, as illustrated in Algorithm 11 regarding any peer $\mathcal{P}$.

Algorithm 11 *join_query_processing_round_robin*()

```
1: while true do
2:   if group G is in Bloom-filter propagation phase then
3:     P gossips the obtained Bloom-filters (if available) to a randomly chosen group member of
       G;
4:   else if the aggregated Bloom-filter is obtained then
5:     P chooses representative P_G^rep through decentralized consensus;
6:     if P hosts the tuples of the bigger partner table then
7:       P checks the aggregated Bloom-filter against its local data;
8:       P sends its tuples that passed the Bloom-filtering test to P_G^rep;
9:     end if
10:  else if P hosts the tuples of the smaller partner table then
11:    P fetches the tuples from P_G^rep and produces intermediate tuples;
12:  end if
13:  repeat
14:    P switches to the next group G that it belongs to;
15:  until G is a group whose query processing is not completed yet
16:  if all the groups P belongs to have completed the query processing then
17:    break; //from the while-loop
18:  end if
19: end while
```

This round-robin fashion algorithm effectively avoids the blocking of any intra-group communication and query processing, while guaranteeing that peers are involved in sufficient (gossip) rounds to complete the processing within each group. The overall number of gossip rounds to complete join query processing is bounded by $\mathcal{O}(c \cdot \ln |G_{max}|)$, where $c \leq s^d$ is the maximum number of the groups that a peer belongs to.

5.3 Join Query Operator and General Query Processing

5.3.1 General Query Plan

It is straightforward to package the two phases into a physical distributed join query operator and integrate it into general query plans. Depending on specific application requirements, the operator

may run the group discovery phase eagerly or passively. For example, when peers join and leave dynamically or data updates are frequent (*e.g.,* as in real-time location-aware services), the group discovery phase may run for each query. Instead, when data distribution is relatively stable such that group membership does not change during a period of time (*e.g.,* as in P2P pub/sub systems), each peer may cache the membership information for potential reuse and issue the group discovery phase periodically.

The realization of the two-phase join query operator is highly modularized, and the operator has correctness guarantee, as proved in Theorem 12. Thus the operator can be seamlessly integrated into general query plans. For example, consider a normal SQL query (in Figure 5.3(a)) that contains range selection operator, join operator and ranking operator (*e.g.,* through order-by statement). Since bandwidth consumption is important for in-network query processing, different optimization heuristics exist for general query plans. For instance, as shown in Figure 5.3(b), the range selection operator is pushed down to the bottom of the query plan tree, which runs locally without incurring communication cost; then the two-phase join operator is executed, followed by the ranking operator.

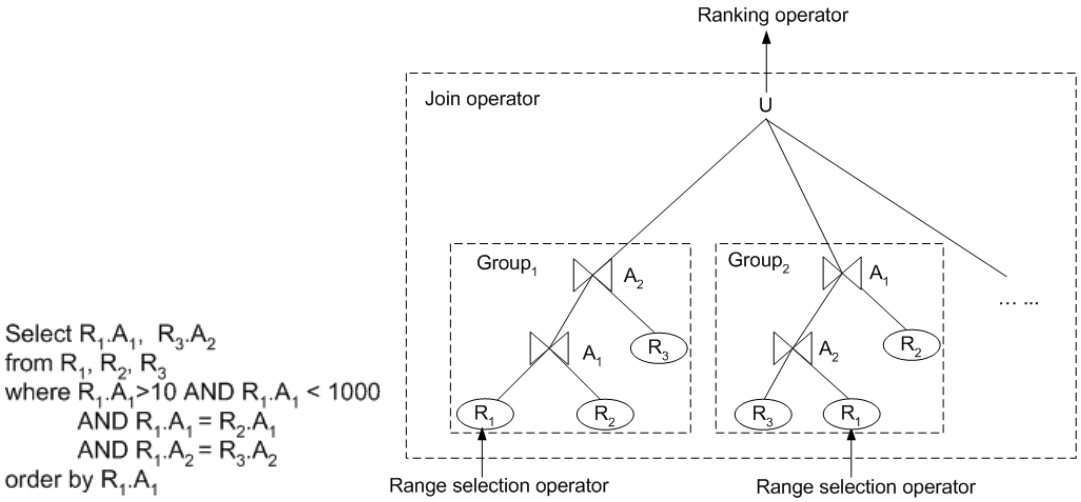


Figure 5.3: SQL Query and its Logical Plan

Recall that the push-sum approach discussed in Section 5.2 can be employed to propagate the cardinality information. The join operand cardinalities over each attribute can be piggybacked during group discovery processing. All peers can independently and consistently decide the order of the join operators without further communications among each other. The ordering can be realized through standard join selectivity computation strategies that have been established in the literature [39]. Depending on the distribution of the tuples within each data space partition, different groups may agree on distinct ordering of the join operators. Since the join query processing of each group is independent to that of all other groups, the distinct orderings do not affect the overall join query processing.

For example, as shown in Figure 5.3(b), Group_1 decides to join R_1 and R_2 first then with R_3, while Group_2 may choose to process R_1 and R_3 over A_2 first then the join operation over

attribute A_1. This enables finer-granularity in-network query optimization. Correspondingly, without sticking to the Bloom join techniques, peers may customize physical query plan for the same query within the group. This does not require communication-intensive coordination among peers. Once all peers obtain consistent information about the selectivity information of join operators, they can agree on the processing order of the join operators and also the physical query plans in a purely decentralized fashion.

5.3.2 Support of Non-equi-join Queries

Recall that peers self-organize into groups based on the partitions at which their projected values over the join attributes are located. By re-defining groups, the proposed approach can be extended to support other non-equal-join types of join queries. For instance, the partition-based approach proposed in this dissertation can be extended to support band-join queries [32]. This has never been achieved by existing approaches (*e.g.,* in PIER [47]) because uniform hashing functions cannot ship the tuples that are relevant to non-equality conditions onto the same peers for join query processing.

A band join query operator joins two partner tables over a join attribute with a range constraint. For instance, given two tables R_1 and R_2 and their join attribute A, a band join may look like $R_1 \bowtie_{R_1.A-c_2 \leq R_2.A \leq R_1.A+c_1} R_2$, where c_1 and c_2 define the boundary of the band.

A straightforward change of the proposed approach can group the peers that host join attribute values that satisfy the band join conditions. This can be achieved by re-defining the group membership according to band-join queries, as detailed below. Recall that, during the group discovery process regarding equi-join predicates (as addressed in Section 5.2.2), each peer obtains the membership of its affiliated group through gossip protocol. Consider any group G, and denote by $\mathcal{P}_G^{rep}$ G's representative through decentralized consensus. Then in addition to the existing group membership of G, $\mathcal{P}_G^{rep}$ will also keep the group membership of G's neighboring groups whose partitions adjoin with G's partition in the data space. Denote by MBB the minimum bounding box that encircles the data points of a given group. Then $\mathcal{P}_G^{rep}$ checks the coordinates of the MBB of group G, denoted by MBB_G, and computes all the neighboring groups (denoted by S) whose MBB coordinates satisfy the given band-join predicate regarding MBB_G. MBB_G's coordinates are then propagated within $G' \in S$. Those peers belonging to G' whose tuples may satisfy the band-join predicate regarding MBB_G's coordinates will reply to $\mathcal{P}_G^{rep}$ its IP addresses. $\mathcal{P}_G^{rep}$ then recruits these peers as new members of G. Since $\mathcal{P}_G^{rep}$ obtains the updated complete membership of group G, it can simply issue a propagation phase to populate the new member information among all G's members. This band-join-aware group discovery process may iterate further across indirect neighboring groups recursively until all the groups whose $MBBs$ intersect with MBB_G are exhausted. Consequently, group G will self-contain all the peers whose data are sufficient to complete band-join queries regarding the partition corresponding to G.

Once all the tuples that may satisfy the band join condition are self-contained by each group. Group-wise band-join query processing can be executed independently to produce the final complete results. Note that, Bloom filters are not suitable here because the Bloom-filtering only resolves exact-matching rather than the range-based predicates. Instead, a semi-join-alike approach is

employed that ships join attribute values to representatives, as illustrated in Algorithm 12. Each peer will still be involved in the band join query processing across multiple groups in a round-robin fashion, similar to Algorithm 11.

Algorithm 12 *Band_join_query_processing*()

1: **if** $\mathcal{P}$'s data belong to a smaller partner table **then**
2: $\mathcal{P}$ propagates the set of projected values (V) over A;
3: **end if**
4: Gossip V among all group members;
5: **if** gossip propagation completes && $\mathcal{P}$'s data belong to the bigger parter table **then**
6: check the values in V against the local projected values over A based on band-join query range predicate;
7: send the intermediate results to the representative peer $\mathcal{P}_G^{rep}$;
8: **end if**
9: **if** $\mathcal{P}$ is the representative peer $\mathcal{P}_G^{rep}$ **then**
10: $\mathcal{P}$ collects intermediate results;
11: $\mathcal{P}$ waits for peers (corresponding to the smaller partner table) to fetch the tuples and join with their data;
12: **end if**

More generally, the band join conditions can be relaxed to capture customized similarity conditions such as $R_1 \bowtie_{dist(R_1.A,R_2.A)<\delta} R_2$, where *dist* denotes a distance function over the join attribute domain and δ represents a similarity threshold that is configurable by users. By making all the tuples that satisfy the similarity condition contained (or retrievable) within each group, group-wise join query processing can be realized in a similar way as addressed for equi-join query processing.

5.4 Performance Evaluation

5.4.1 Experimental Setting

The p2psim discrete event simulator is employed to simulate P2P networks with up to 4000 peers. A two-dimensional grid space of length 100 milliseconds at each side is generated, and peers are uniformly distributed within the space. The latency between peers is computed as the Euclidean-distance between the corresponding coordinates within the space.

TPC-H benchmark is employed to generate data load since it supplies multi-join queries. The benchmark data generator (*i.e., dbgen*) is run with scale factor set to 1, creating a database of eight relational tables. Given a join query, each peer randomly chooses a table from the database in the following way: (1) the schema of the chosen table includes the join attributes of the query; (2) each table T in the database is chosen randomly with a weighted probability proportional to the number (denoted by $|T|$) of tuples in T; (3) once a table (*e.g., T*) is chosen, the number of tuples to be

obtained by each peer equals $min\{150,\ 0.1\times|T|\}$; with respect to a network of 4000 peers, the overall number of tuples is 599,701[5], which is sufficient to simulate the join query processing of various approaches; and (4) the obtained tuples by each peer follow the data distribution mechanisms to be described shortly. Although each peer hosts a horizontal fragment of a single table in this simulation, the proposed approach can be easily extended to allow each peer managing data from multiple tables.

Two commonly-used data distribution schemes are considered: *uniform random distribution* and *skewed distribution*. Under the former, each peer simply chooses tuples from the corresponding table by following uniform random distribution. However, in real applications such as location-aware distributed services, geographically proximate data are often clustered on peers; thus the skewed data distribution is considered: each peer chooses an initial tuple (uniform) randomly from its chosen table and obtains a specified number of nearest neighbors of the initial tuple (with respect to the join attributes) from the same table.

```
Select ORDERDATE, ORDERS-PRIORITY
from lineitem, orders
where lineitem.ORDERKEY = orders.ORDERKEY
```

(a) q_1

```
Select ACCTBAL, NAME, ADDRESS, PHONE,
       COMMENT, MFGR, PARTKEY
from supplier, partsupp, part
where supplier.SUPPKEY = partsupp.SUPPKEY
      and partsupp.PARTKEY = part.PARTKEY
```

(b) q_2

```
Select ACCTBAL, NAME, ADDRESS, PHONE,
       COMMENT, NAME, MFGR, PARTKEY
from supplier, partsupp, part, nation
where supplier.SUPPKEY = partsupp.SUPPKEY
      and partsupp.PARTKEY = part.PARTKEY
      and nation.NATIONKEY = supplier.NATIONKEY
```

(c) q_3

```
Select ACCTBAL, NAME, ADDRESS, PHONE,
       COMMENT, NAME, MFGR, PARTKEY
from supplier, partsupp, part, nation, region
where supplier.SUPPKEY = partsupp.SUPPKEY
      and partsupp.PARTKEY = part.PARTKEY
      and nation.NATIONKEY = supplier.NATIONKEY
      and region.REGIONKEY = nation.REGIONKEY
```

(d) q_4

Figure 5.4: Join Queries

Four join queries with up to four join attributes are evaluated, as shown in Figure 5.4. The two-way and five-way join queries (*i.e.,* Figure 5.4(a)(c)) are transcribed from the queries supplied by the TPC-H benchmark, while the three-way and four-way join queries (*i.e.,* Figure 5.4(b)(d)) are manually created since they are not provided by the benchmark. Low-dimensionality is considered in this work (*e.g.,* $d \leq 5$), which is sufficient for many practical applications. For example, location-aware services in mobile computing systems may require only a few join attributes (*e.g.,* the longitude and latitude corresponding to locations). For higher dimensionality (*e.g.,* tens or hundreds of features as in images and video data), a feasible solution may be to use dimension-reduction techniques [76].

For comparison purposes, the distributed hash join approach and its variants [47] are implemented under the p2psim as the baseline approaches, including distributed hash join approach, symmetric semi-join approach, and Bloom-join approach. Details of these approaches have been

[5]With TPC-H, the total data volume in this simulation is over 76 MB, which is relatively small but does not affect the validity of the evaluation because each peer stores sufficient tuples and tuple size decides the overall data volume.

presented in Chapter 2.

Chord protocol [87] is chosen as the underlying DHT-based routing mechanism, which is supplied by p2psim. Due to lack of data distribution information, the order of join attributes is randomly chosen and all join query operators are conducted in a non-blocking fashion.

5.4.2 Evaluation of Group Construction and Maintenance Cost

In the group discovery and construction process, peers join the groups corresponding to the partitions that their hosted data reside in. A comparison of the bandwidth cost and processing time of both the proposed grouping process and the construction of DHT substrate (*i.e.,* Chord) with network size $N = 4000$ under two data distributions is shown in Table 5.1. Without loss of generality, the data space for query q_4 is considered, covering all four join attributes considered in this performance evaluation (in Figure 5.4). Since the group discovery and construction processes rely on uniform gossip mechanism, the bandwidth cost is slightly higher than that incurred by DHT routing infrastructure. However, the group construction latency is significantly less than that incurred by the DHT routing infrastructure. This is because all peers execute group construction process in parallel, while the cost of DHT construction is proportional to the number of peers in the network. The grouping process needs to run initially only once, such that the group construction cost is amortized over the whole query processing period, leading to a potentially small overhead per query that is inversely proportional to the number of queries.

Cost	Bandwidth(kB)	Latency(ms)
Grouping Construction ($uniform$)	5294.68	3468
Grouping Construction ($skewed$)	1263.68	1020
DHT Construction	3839.04	2039490

Table 5.1: Infrastructure Construction Cost

When data change on peers, the group membership may change. Similarly, during network churn, the membership information of all the involved groups needs to be updated. In Table 5.2, the group maintenance cost (including bandwidth cost and response time) during network churn is compared against that of the DHT routing information updates. Specifically, the network churn rate is set as $p = 0.1$, which indicates that, 10% of the peers join or leave the network at any moment of the evaluation time. This potentially incurs the membership information updates across multiple groups. For simplicity, an equivalent number of peers join the network in the meantime to keep the network size stable during the whole evaluation.

The results demonstrate that the latency of the gossip-based approach under both uniform and skewed data distributions is lower than that needed by the DHT infrastructure. However, the bandwidth cost of the group maintenance under uniform data distribution may be higher than that incurred by DHT overlays, because each peer may span a larger number of groups than that under skewed data distribution, taking more gossip messages (and rounds) to complete the maintenance. The results also reflect the maintenance cost of data changes, which incur similar membership

Cost	Bandwidth(kB)	Latency(ms)
Group maintenance (*uniform*)	39.19	124
Group maintenance (*skewed*)	0.52	114
DHT Maintenance	3.432	610

Table 5.2: Infrastructure Maintenance Cost

change of multiple groups.

Both the group construction and maintenance costs are orthogonal to the query processing cost to be measured next. Moreover, the amortized construction cost and maintenance overhead (*e.g.,* when $p = 0.1$) are insignificant in comparison to the query processing cost. Thus they will not be discussed in the remainder of this section.

5.4.3 Evaluation of Query Processing Performance

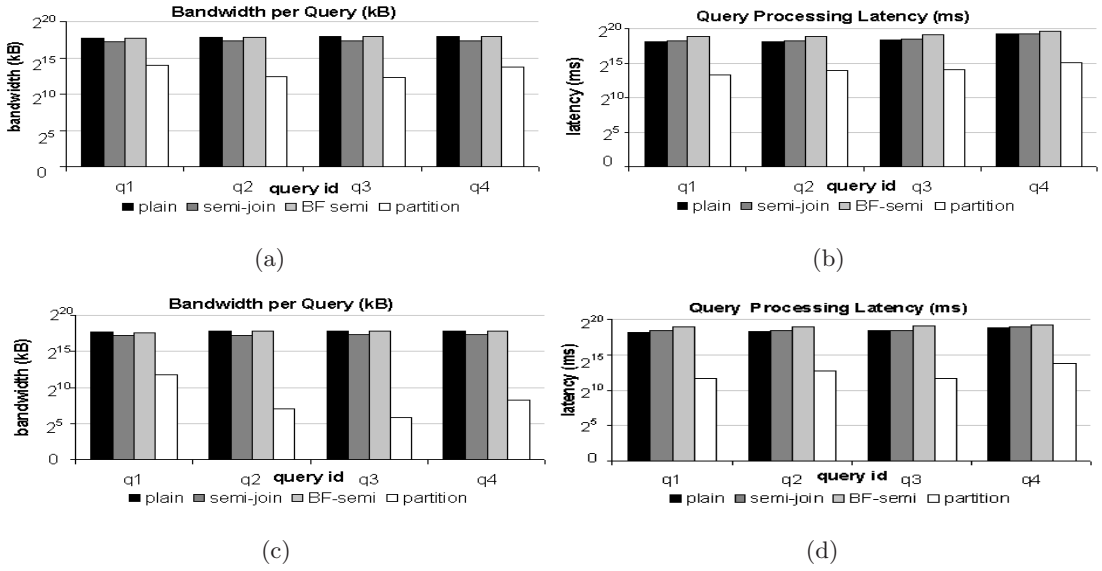


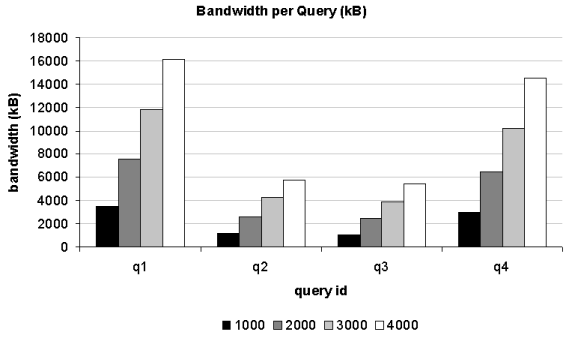
Figure 5.5: Query Processing Cost under Various Data Distribution Schemes

Bandwidth consumption and query execution latency under the uniform random and skewed data distribution schemes are evaluated. Without loss of generality, the partition number over each join attribute dimension is set to be 3. This leads to 3 partitions for q_1 and 81 partitions for q_4, which is sufficient to test the behavior of the proposed approach under a number of data space partitions. The sensitivity of the query processing performance over other numbers of partitions will be addressed shortly in Section 5.4.5. For presentation, in the following figures, the proposed approach is referred to as "partition"; the distributed hash join approach is referred to as "plain"; the symmetric semi-join approach is referred to as "semi-join", and the Bloom-join approach is referred to as "BF-semi".

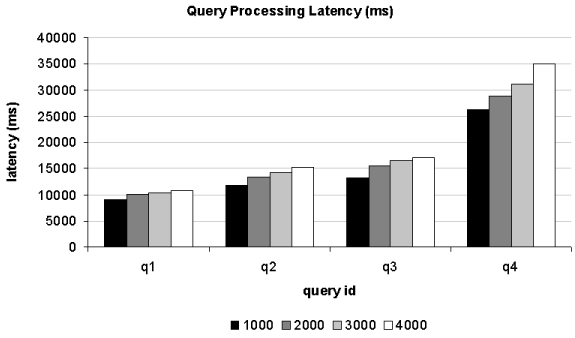
Under uniform random data distribution, the bandwidth consumption of join query processing is shown in Figure 5.5(a), which illustrates that the proposed approach is more efficient than the baseline approaches. The results are shown in logarithmic-scale to ease the presentation. Figure 5.5(b) demonstrates the latency of the query processing including local processing cost. The results show that the proposed approach take less time to complete due to the exploitation of parallelism and in-advance pruning of irrelevant peers from query processing. Without loss of generality, the latency of processing each tuple is assumed to be 1 millisecond and each peer processes all tuples sequentially.

Similar experimental results are obtained under the skewed data distribution, as shown in Figures 5.5(c)(d). In comparison to the uniform random data distribution setting, both bandwidth and query processing latency of the proposed approach are much lower because the data that are proximate over join attributes are clustered on a subset of peers in the network such that both the number of groups and the group cardinalities may be much smaller, lowering query processing cost even further.

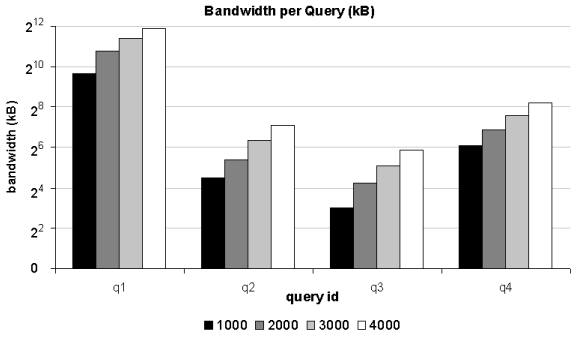
5.4.4 Evaluation of Scalability



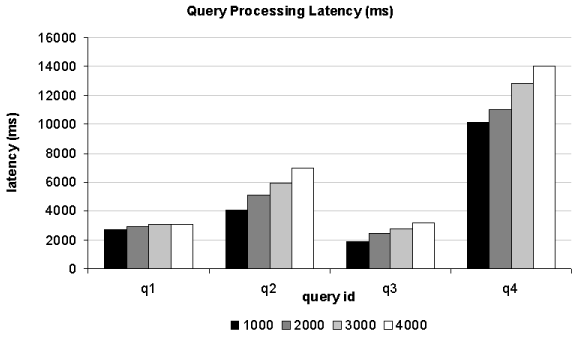
(a) bandwidth under uniform distribution



(b) latency under uniform distribution



(c) bandwidth under skewed distribution



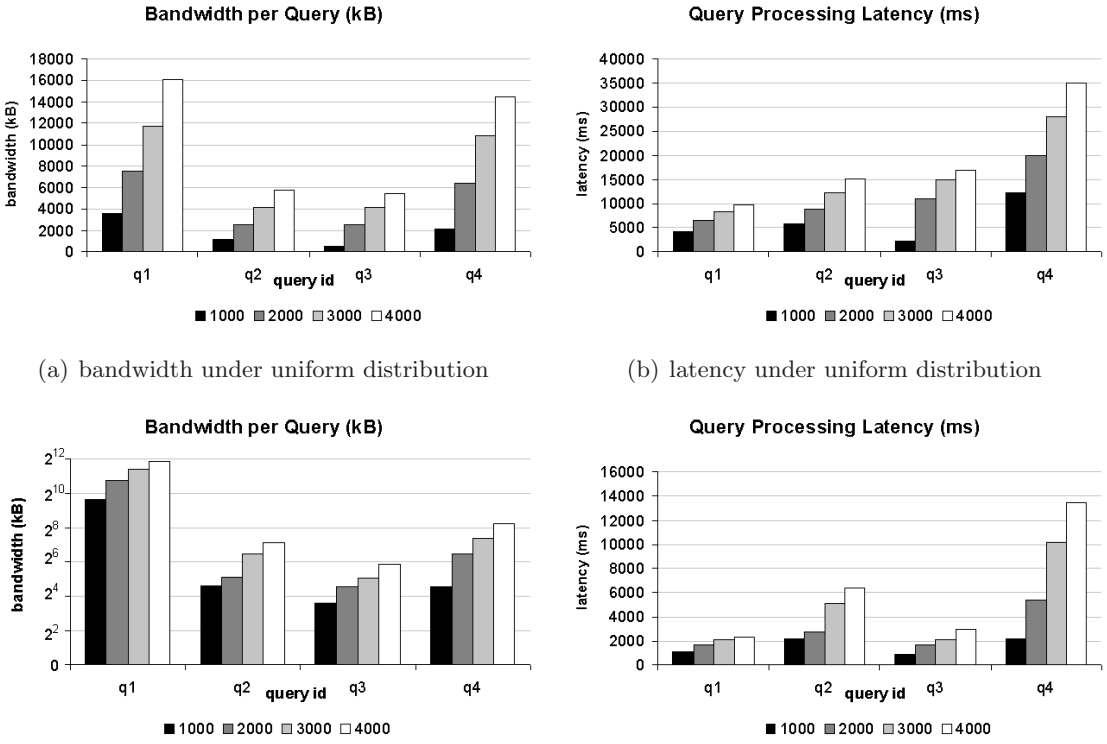
(d) latency under skewed distribution

Figure 5.6: Scalability Test with Increasing Data Volume

Both the increase of data volumes on peers and the scale-up of the network itself affect query processing performance. Scalability tests are conducted under the following settings.

The first setting fixes the network size (*i.e.,* 4000) while making the total data volume proportional to the network size. The query processing bandwidth and latency under uniform random distribution and skewed distribution are demonstrated in Figure 5.6 respectively. Both metrics increase correspondingly because more tuples are shipped among peers and are processed locally.

Second, network size increases and peers receive relational tuples under uniform and skewed data distributions. In other words, in the simulation, each peer randomly selects a relational table and chooses up to 150 tuples under different data distribution schemes. The query processing bandwidth and latency under uniform random distribution and skewed distribution are demonstrated in Figure 5.7 respectively. The results show that, both the bandwidth and query processing latency grow steadily, due to the increase of both network size and data volume.



(a) bandwidth under uniform distribution

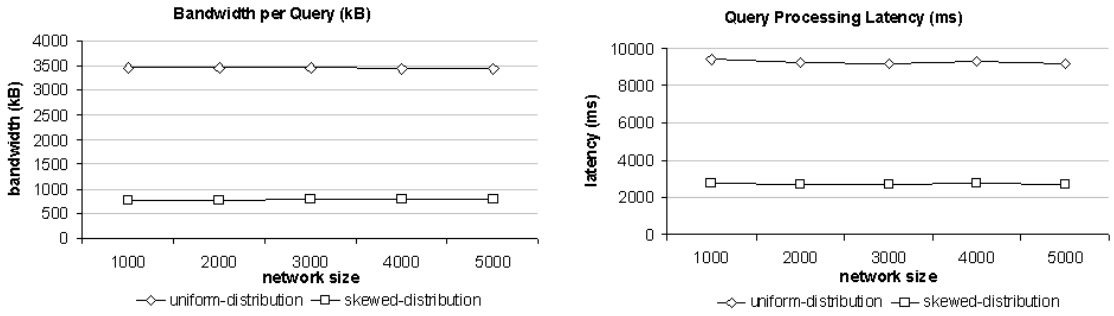
(b) latency under uniform distribution

(c) bandwidth under skewed distribution

(d) latency under skewed distribution

Figure 5.7: Scalability Test with Increasing Network Size and Data Volume

Finally, performance evaluation is also conducted for the case that data volume does not change while network size increases. Since the partition-based approach proposed in this dissertation depends on the grouping of peers regarding join query processing, its performance relies on the volume of the data that are involved in the join query processing. Thus the approach is not sensitive to the mere network size increase. The join query processing bandwidth and latency of query q_1 is shown in Figure 5.8. Similar results are obtained for other queries as well.



(a) (b)

Figure 5.8: Scalability Test with Increasing Network Size

5.4.5 Evaluation of Sensitivity

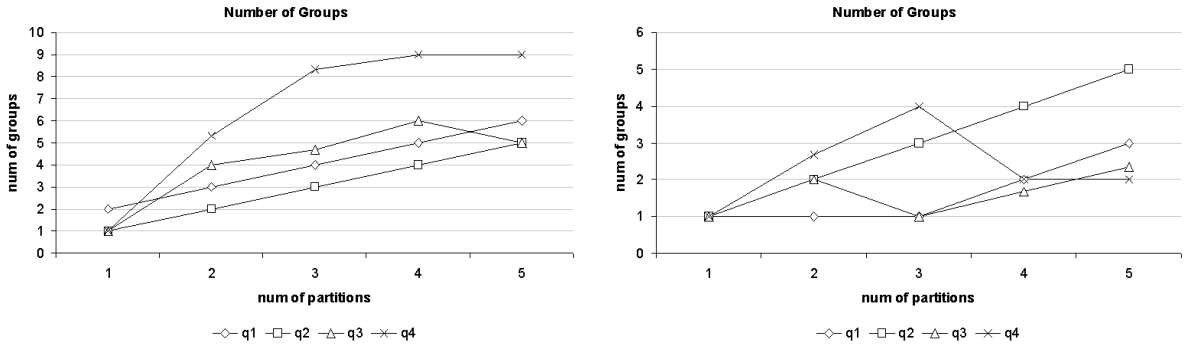
The bandwidth consumption and query processing latency (as shown in Figure 5.10) over different number of partitions (*i.e.,* 1, 2, 3, 4 and 5) are evaluated under both uniform and skewed data distribution. When the number of partitions increases, the query processing cost (local processing cost inclusive) may not always decrease monotonically. This is because the partitioning of the data space is oblivious to data distribution. Thus the number of groups may not increase when the number of partitions is higher, as demonstrated by the correlation study between the number of groups and the number of partitions in Figure 5.9.

Figure 5.10(a)(d) show that, the bandwidth cost may decrease even when the number of partitions goes up. This is because, under specific data distribution, peers may form fewer groups when the number of partitions goes up, leading to fewer number of gossip messages and bandwidth overhead. In Figure 5.10(b)(e), the query processing latency under round-robin fashion (Algorithm 11) increases sharply because of the increase of the number of gossip rounds. In contrast, Figure 5.10(c)(f) show that the "concurrent" query processing latency is insensitive to the number of partitions, when all group-wise gossiping is executed in parallel without round-robin fashion. This is because the gossiping cost is decided by group size now, instead of by either the number of partitions or the number of groups.

5.5 Summary

In massively distributed P2P networks, the integration of multiple data sources is critical. In this chapter, the problem is addressed within the framework of distributed relational join query processing and an effective approach is developed to enable parallel join query processing with performance guarantees. Novel and scalable mechanisms are addressed to explore membership information for groups, each corresponding to a disjoint partition in the data space over join attributes. Each group runs bandwidth-efficient Bloom join query operations to produce group-wise join query results, which are eventually aggregated to produce final results.

Through rigorous analysis and extensive experiments, the proposed approach is proved effective

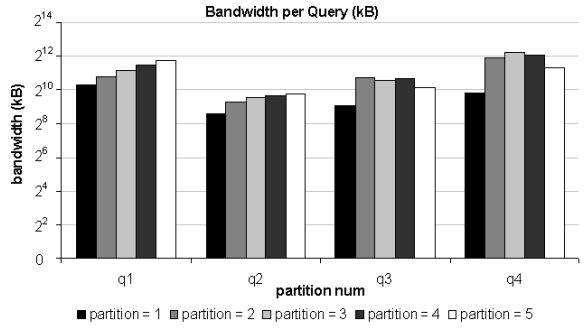


(a) Uniform Data Distribution

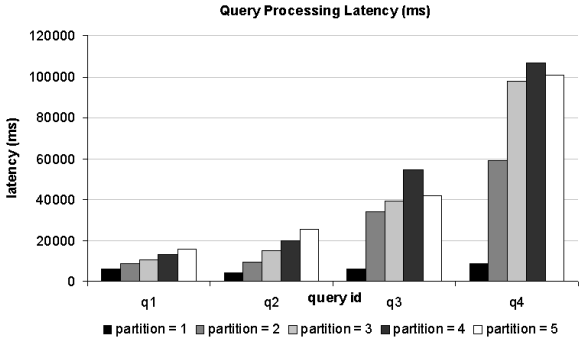
(b) Skewed Data Distribution

Figure 5.9: Correlation between Number of Partitions and Groups

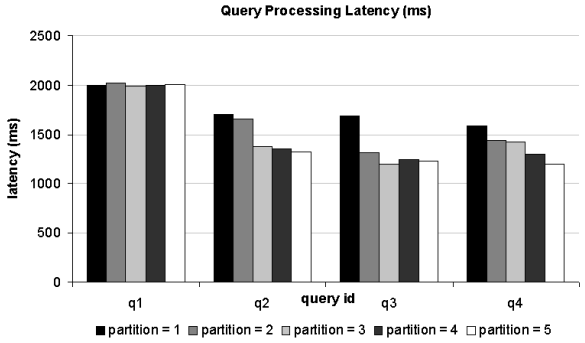
and efficient. Due to guaranteed correctness, the proposed join query operator can be easily integrated in general query plans, enhancing its viability in various application environments. Moreover, the proposed approach can be easily extended to support non-equi join queries such as band join queries.



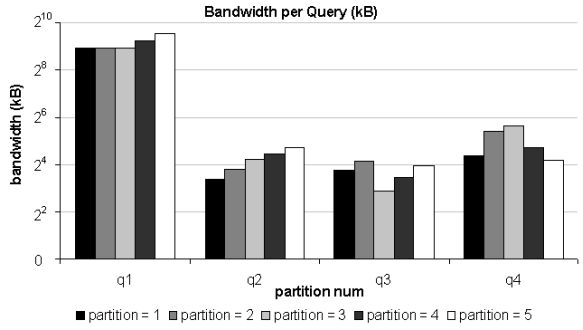
(a) bandwidth under uniform distribution



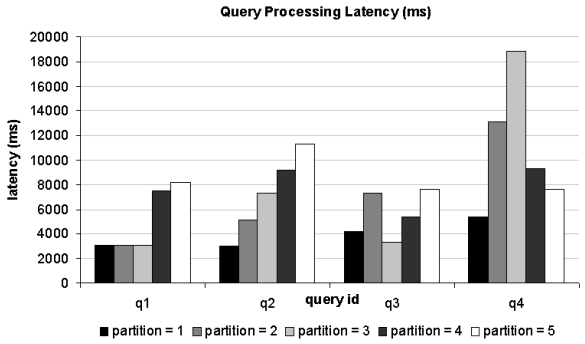
(b) latency under uniform distribution



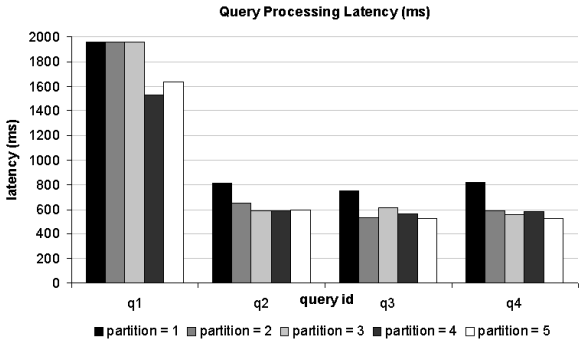
(c) concurrent latency under uniform distribution



(d) bandwidth under skewed distribution



(e) latency under skewed distribution



(f) concurrent latency under skewed distribution

Figure 5.10: Evaluation of Sensitivity

Chapter 6

Conclusion

6.1 Summary of Contributions

P2P infrastructure is an important architecture to support massively distributed large-scale applications. Complex query operators such as range operator and join operator are essential for various applications such as content distribution, distributed spatio-temporal services, data integration, and many others. Although these operators have been widely studied in the literature, multiple open research problems exist, both important and challenging to solve.

First, tree-based structured P2P overlay networks have been developed to support efficient range query processing (*e.g.,* BATON, P-Tree, PHT and others). However, existing approaches usually disregard the impact of (high) network churns over query processing performance. Moreover, these overlay networks lack configurability to supply strong performance guarantees independent of network churn rates, leading to performance failure with respect to fault-tolerance specifications (*e.g.,* a specified number of hops or latency per query). In this dissertation, a B-tree-based configurable overlay network is addressed (*i.e.,* VOILA) that can adjust overlay network structure and routing algorithm according to network churn rates so as to provide strong performance guarantees independent of varying network churn rates. Alternatively, without making significant changes to overlay network structures, effective replication schemes are proposed at the query processing level to achieve the same goal for those tree-based overlay networks that lack configurability. Experimental results show that, the replication schemes that are proposed at both the overlay network and the query processing levels effectively provide strong performance guarantees with respect to fault-tolerance specifications, providing a suitable solution to those applications that require strict performance guarantees (*e.g.,* on-demand P2P video sharing systems).

Second, range query processing under unstructured P2P architecture is important to support query processing over range data (*e.g.,* video clips that are characterized with timestamp intervals) or to introduce new functionalities over point data (*e.g.,* search song files that are produced within a specified period). Without imposing indexes over distributed data, unstructured P2P architectures usually employ flooding, gossip, or random walk mechanism to ship queries. Query

results are cached at multiple peers to improve query processing performance (*e.g.,* the number of hops or latency per query). With respect to range query processing, cached data items constitute distributed range caches that can be exploited for subsequent query processing. Since range queries usually request multiple (distinct) data items, the query processing performance may be dominated by the retrieval cost of those data items that have not been well cached (*i.e.,* poorly-replicated data items) within the network. In this dissertation, an efficient approach is devised that facilitates the query processing by prefetching poorly-replicated data items. Prefetching techniques have already been widely studied. The proposed approach is novel in that it is popularity-aware, which improves the adaptivity of prefetching and is expected to be more cost-effective (with respect to bandwidth consumption and query processing latency) than simply prefetching all correlated data items. Theoretical analysis and experiments show that the proposed approach substantially improves query execution performance.

Finally, in unstructured P2P networks, heterogeneous data items are shared across multiple peers and join queries are essential in supporting data integration. Two important design principles include: decentralized catalog management and efficient query processing. Departing from existing approaches that depend on either fully-replicated data placement information or DHT-based mechanisms, a partition-based parallel join query processing approach is proposed in this dissertation. By clustering peers within groups such that each group conducts join query processing independently, intra-operator parallelism is enabled. Simulation shows that the proposed approach achieves better performance against comparison systems with respect to bandwidth cost and query processing latency. Moreover, since the correctness of join query processing is guaranteed, the proposed join query operator can be easily applied within general query processing plans, enhancing its viability in various application environments. Performance evaluation demonstrates that the proposed approach takes significantly less bandwidth and latency to process join queries than the baseline approaches that have been proposed in the literature.

6.2 Potential Applications

In addition to the motivating examples that have been presented, the proposed approaches are suitable for the following potential applications under P2P overlay network architectures.

6.2.1 Complex Query Processing across Data Centers

Nowadays, large-scale computing, especially complex query processing, is often handled by clusters of computers grouped in data centers that include hundreds or thousands of computing nodes. Large organizations locate these clusters at data centers distributed around the globe. Examples of such systems include the computing backends of Web sites such as Google, eBay, and Amazon. Although there are many different deployments, they share the following characteristics: (1) the computing nodes are commodity hardware with few constraints on their storage and processing capabilities; (2) nodes communicate with each other in P2P mode for query processing through the underlying Internet; (3) communication among nodes within the same data center is much cheaper

than that across data centers; and (4) failures (*e.g.,* network failure, node failure, or power outage) cause nodes to go down and come back up at arbitrary time.

VOILA, the tree-based overlay network developed in this dissertation, can be employed to support complex queries (*e.g.,* range queries) across distributed data centers. Thus the replication schemes proposed in this dissertation (Chapter 3) can be directly employed to realize range query processing with strong performance guarantees.

In addition to direct range query processing over distributed data centers, another potential application that fits the data center scenario is autonomic computing resource sharing. Computing resource may include storage, CPU capacity, network bandwidth and so on. Once the quota of a specific resource is represented as range data items, the range query processing approaches in this dissertation can support the search and management of these computing resources in purely decentralized fashion.

6.2.2 Mobile P2P Computing

CarTel system [19] employs sensors installed on cars to monitor traffic and road conditions in real-time fashion. The system employs centralized servers to maintain and analyze the information collected by cars on the road, which may potentially pose scalability and robustness problems.

The decentralized tree-based overlay network that is studied in this dissertation is suitable to support query processing (*e.g.,* the range query processing techniques proposed in Chapter 3) for such mobile computing infrastructure: (1) peers are organized without employing centralized mechanisms, alleviating scalability and robustness problems, meanwhile reducing the administrative tasks that are necessary for centralized mechanisms; (2) tree-based overlay networks provide a scalable query routing mechanism with guaranteed performance (*e.g.,* the number of hops or latency per query); and (3) even under network churn, which is common in mobile computing systems, the replication schemes proposed in this dissertation can supply sufficient configurability for strong performance guarantees with respect to fault-tolerance specifications.

6.2.3 Range-aware P2P Content Search and Storage

Existing file sharing systems usually use file names to identify data content (*e.g.,* song files) that are distributed in P2P networks. However, future content distribution systems may require the support of range-based data and impose range constraints over queries. For example, timestamp interval of cached video clips are considered as range data items, and range queries that cover specific timestamp intervals can be issued to locate all the video clips that are playbacked at the query issuer site.

Moreover, in a P2P storage sharing system, peers may claim a certain amount of free disk space, which are modeled as range data since the disk space may change dynamically. When any peer asks for storage space, it may issue a range query, which returns candidate peers that can satisfy the storage requirements.

When these applications are deployed under unstructured P2P architecture, the approach

developed in Chapter 4 can be directly applied to improve query processing performance.

6.2.4 Data Integration across Multiple P2P Networks

Multiple P2P networks exist that handle different data contents. For example, Minerva system [12, 66] employs peers to crawl and maintain Web data. Instead of posing Web search queries to centralized servers, peers ship queries to the distributed peers that host matching results through routing mechanisms, improving the scalability of the system. Differently, file sharing P2P networks such as Gnutella may handle distributed multimedia contents. Since these overlay networks manage data content with different schema and semantics, the integration of them produce interesting mashups. For example, a mashup system across the two P2P networks may enhance the Web pages that describe songs or music with hyper-links pointing to the downloadable files available in Gnutella. Such a data integration functionality can be realized based on the join query processing approach proposed in Chapter 5.

6.3 Future Research Directions

6.3.1 DHT-based P2P Join Query Processing

DHTs have been widely studied in literature and have been put into practical use (*e.g.,* the Kad DHT tracker in BitTorrent[1]). Complex query processing approaches over DHT have been developed. In PIER, multiple schemes are proposed to support efficient DHT-based join query processing, as discussed in Section 2.2. However, as indicated in Chapter 5, oblivious to data distribution over join attributes, these approaches may incur high bandwidth cost.

There are several open problems on how to improve join query processing performance over DHTs. First, it is unclear how to exploit join index information in facilitating the query processing. Second, how to employ peers to manage the join index information, and decide the number of such indexing peers with respect to peer capacity and workload. Third, although query processing optimization mechanisms have been extensively studied for traditional distributed architectures (*e.g.,* client-server architecture), how to develop optimization strategies under P2P architecture is nontrivial. For example, regarding multi-way join queries that involve multiple join attributes, the selectivity of the attributes needs to be measured and maintained over DHT to decide the order of the join attributes.

6.3.2 Trust-aware P2P Query Processing

Security issue is important for many applications that are deployed in P2P networks, where peers' behavior is hard to control. To guarantee the quality of the data that are shared by peers, trust mechanism can be introduced in the system. This can be modeled as trust-aware query processing problem, which can potentially be tackled with the following solution: a confidence score can

[1] http://www.bittorrent.com

be attached to peers or the shared content; then a trust mechanism can be built to identify the confidence value in a decentralized fashion.

The challenge of this solution is on how to obtain the confidence scores and how to define the trustiness of peers (or data) based on the scores. The confidence scoring may be evolved through social networks (*e.g.,* orkut[2]). A similar idea has been exploited in [63], which however is focused on specific file sharing applications over DHTs. Moreover, it is nontrivial to update the scores in the runtime to reflect the changes of peer behaviors.

6.3.3 P2P Replication and Data Consistency

Data replication mechanism is studied and employed in this dissertation to enhance the fault-tolerance of the system (Section 3.3) and to improve query processing performance (Section 4.3).

Depending on applications, different mechanisms to achieve data consistency can be employed. For example, without deploying distributed locking mechanism (*e.g.,* Chubby [18]), gossip protocol can be used to propagate data updates among peers for "eventual consistency". A direction of the future work is to study how to provide data consistency under different semantics (*e.g.,* strong and weak data consistency) and how to materialize these semantics in the approaches that have been developed in this dissertation.

6.4 Closing

As a modern distributed computing infrastructure, P2P overlay networks provide a disruptive technology that employs abundant computing resources and data contents at the edge of the network to support large-scale applications in decentralized, scalable, and efficient fashion [73]. Although distributed data management problems have been well studied under client-server architecture [74], the modern P2P architectures pose new challenges due to its large-scale, dynamism, and other characteristics. Complex query operators including range and join query operators are studied in this dissertation to facilitate the distributed data management and query processing under various P2P architectures.

The research on P2P data management and query processing has attracted considerable attention from the academia. It is also noted recently that the industry has started to actively employ commodity hardware in P2P fashion to support large-scale distributed computing tasks. Distributed computing software development platforms have been developed (*e.g.,* Dynamo [30]). Such platforms will facilitate the realization of modern large-scale applications, and effectively power the complex query operators for next-generation distributed computing.

[2] http://www.orkut.com

Bibliography

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proc. Sixth International Conference on Cooperative Information Systems*, 2001.

[2] K. Aberer. Efficient search in unbalanced, randomized peer-to-peer search trees. Technical Report IC/2002/79, EPFL, 2002.

[3] Ioannis Aekaterinidis and Peter Triantafillou. Substring matching in P2P publish/subscribe data management networks. In *Proc. Int. Conf. on Data Engineering*, pages 1390–1394, 2007.

[4] A Allavena and S. Keshav. Fast, efficient and robust in-network computation. Technical Report cs-2006-22, University of Waterloo, 2006.

[5] A. Allavena, Q. Wang, I. Ilyas, and S. Keshav. LOT: A robust overlay for distributed range query processing. Technical Report cs-2006-21, University of Waterloo, 2006.

[6] Emmanuelle Anceaume, Maria Gradinariu, Ajoy Kumar Datta, Gwendal Simon, and Antonino Virgillito. A semantic overlay for self-* peer-to-peer publish/subscribe. In *Proc. 26th Int. Conf. on Distributed Computing Systems*, page 22, 2006.

[7] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Peer-to-Peer Computing*, 2002.

[8] J. Aspnes and G. Shah. Skip graphs. In *Proc. of the 2003 ACM-SIAM Symp. on Discrete Algorithms*, pages 384–393, 2003.

[9] E. Babb. Implementing a relational database by means of specialized hardware. *ACM Trans. Database Syst.*, 4(1):1–29, 1979.

[10] W. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. In *Proc. Int. Conf. on Data Engineering*, pages 174–185, 2005.

[11] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[12] M. Bender, S. Michel, G. Weikum, and C. Zimmer. The MINERVA project: Database selection in the context of P2P search. In *Datenbanksysteme in Business, Technologie und Web*, pages 125–144, 2005.

[13] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie. Query processing in a system for distributed databases. *ACM Trans. Database Syst.*, 6(4):602–625, 1981.

[14] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proc. ACM SIGCOMM*, 2004.

[15] Kenneth P. Birman. *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer, 2005.

[16] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[17] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[18] Michael Burrows. The chubby lock service for loosely-coupled distributed systems. In *7th Symposium on Operating Systems Design and Implementation*, pages 335–350, 2006.

[19] Vladimir Bychkovsky, Kevin Chen, Michel Goraczko, Hongyi Hu, Bret Hull, Allen Miu, Eugene Shih, Yang Zhang, Hari Balakrishnan, and Samuel Madden. Data management in the CarTel mobile sensor computing system. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 730–732, 2006.

[20] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002.

[21] Bin Cheng, Xuezheng Liu, Zheng Zhang, and Hai Jin. A measurement study of a peer-to-peer video-on-demand system. In *Peer-to-Peer Systems, First International Workshop*, 2007.

[22] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *Proc. 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*.

[23] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. ACM SIGCOMM*, pages 177–190, 2002.

[24] John H. Conway and Richard Guy. *The Book of Numbers*. Springer, 1998.

[25] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. Querying peer-to-peer networks using P-Trees. In *Proc. 7th Int. Workshop on the World Wide Web and Databases (WebDB)*, pages 25–30, 2004.

[26] Adina Crainiceanu, Prakash Linga, Ashwin Machanavajjhala, Johannes Gehrke, and Jayavel Shanmugasundaram. P-Ring: an efficient and robust P2P range index structure. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 223–234, 2007.

[27] F. Dabek, J. Li, E. Sit, J. Robertson, M. Frans Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput. In *1st Symposium on Networked Systems Design and Implementation*, pages 85–98, 2004.

[28] Frank Dabek. A cooperative file system. Master's thesis, Massachusetts Institute of Technology, 2001.

[29] Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl Aberer. Range queries in trie-structured overlays. In *Peer-to-Peer Computing 2005*, pages 57–66, 2005.

[30] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Proc. 21th ACM Symp. on Operating System Principles*, pages 205–220, 2007.

[31] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. E. Sturgis, D. C. Swinehart, and D. B. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. ACM SIGACT-SIGOPS 6th Symp. on the Principles of Dist. Comp.*, pages 1–12, 1987.

[32] D. J. DeWitt, J. F. Naughton, and D. A. Schneider. An evaluation of non-equijoin algorithms. In *Proc. 17th Int. Conf. on Very Large Data Bases*, pages 443–452, 1991.

[33] Shantanu Dutt and John P. Hayes. On designing and reconfiguring k-fault-tolerant tree architectures. *IEEE Trans. Computers*, 39(4):490–503, 1990.

[34] H. M. Edwards. *Riemann's Zeta Function*. Academic Press, 1974.

[35] L. G. Erice, E. Biersack, P. Felber, K. W. Ross, and G. U. Keller. Hierarchical peer-to-peer systems. In *Euro-Par 2003. Parallel Processing, 9th International Euro-Par Conference*, pages 1230–1239, 2003.

[36] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Randomized broadcast in networks. *Random Structures and Algorithms*, 1:447–460, 1990.

[37] W. Fontijn and P. A. Boncz. AmbientDB: P2P data management middleware for ambient intelligence. In *Proc. Workshop on Middleware Support for Pervasive Computing (PerWare)*, pages 203–207, 2004.

[38] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Processing queries in a large peer-to-peer system. In *Proc. of the 15th Int. Conf. on Advanced Information Systems Engineering*, 2003.

[39] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database System Implementation*. Prentice-Hall, 2000.

[40] Sarunas Girdzijauskas, Anwitaman Datta, and Karl Aberer. Oscar: A data-oriented overlay for heterogeneous environments. In *Proc. Int. Conf. on Data Engineering*, pages 1365–1367, 2007.

[41] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proc. 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, 2004.

[42] S. D. Gribble, A. Y. Halevy, Z. G. Ives amd M. Rodrig, and D. Suciu. What can database do for peer-to-peer? In *Proc. 4th Int. Workshop on the World Wide Web and Databases (WebDB)*, pages 31–36, 2001.

[43] A. Gupta, D. Agrawal, and A. E. Abbadi. Approximate range selection queries in peer-to-peer systems. In *First Biennial Conference on Innovative Data Systems Research*, 2003.

[44] N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[45] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[46] R. Huebsch, B. N. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of PIER: an Internet-scale query processor. In *CIDR 2005, Second Biennial Conference on Innovative Data Systems Research*, pages 28–43, 2005.

[47] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the internet with PIER. In *Proc. 29th Int. Conf. on Very Large Data Bases*, pages 321–332, 2003.

[48] H.V.Jagadish, B. C. Ooi, K. L. Tan, Q. H. Vu, and R. Zhang. Speeding up search in peer-to-peer networks with a multi-way tree structure. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2006.

[49] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational datatbases. In *Proc. 29th Int. Conf. on Very Large Data Bases*, pages 754–765, 2003.

[50] Sitaram Iyer, Antony I. T. Rowstron, and Peter Druschel. Squirrel: a decentralized peer-to-peer web cache. In *Proc. ACM SIGACT-SIGOPS Symp. on Principles of Dist. Comp.*, pages 213–222, 2002.

[51] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. BATON: A balanced tree structure for peer-to-peer networks. In *Proc. 31th Int. Conf. on Very Large Data Bases*, 2005.

[52] H. V. Jagadish, B.C. Ooi, Q.H. Vu, and A.Y. Zhou R. Zhang. VBI-tree: a peer-to-peer framework for supporting multi-dimensional indexing schemes. In *Proc. Int. Conf. on Data Engineering*, 2005.

[53] M. Jelasity, R. Guerraoui, A. Kermarrec, and M. Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Middleware 2004, IFIP/ACM International Conference on Distributed Systems Platforms*, pages 79–98, 2004.

[54] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3), 2007.

[55] S. Kashyap, S. Deb, K.V.M. Naidu, R. Rastogi, and A. Srinivasan. Efficient gossip-based aggregate computation. In *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 308–317, 2006.

[56] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 482–491, 2003.

[57] G. Koloniari, Y. Petrakis, and E. Pitoura. Content-based overlay networks for XML peers based on multi-level bloom filters. In *Proc. First International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, 2003.

[58] Dongsheng Li, Xicheng Lu, Baosheng Wang, Jinshu Su, Jiannong Cao, Keith C. C. Chan, and Hong Va Leong. Delay-bounded range queries in DHT-based peer-to-peer systems. In *Proc. 26th Int. Conf. on Distributed Computing Systems*, page 64, 2006.

[59] M. Li, W. Lee, and A. Sivasubramaniam. Semantic small world: An overlay network for peer-to-peer search. In *12th IEEE International Conference on Network Protocols*, pages 228–238, 2004.

[60] M. Lin, K. Marzullo, and S. Masini. Gossip versus deterministically constrained flooding on small networks. In *Distributed Computing, 14th International Conference*, pages 253–267, 2000.

[61] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative routing: extensible routing with declarative queries. In *Proc. ACM SIGCOMM*, pages 289–300, 2005.

[62] L. F. Mackert and G. M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *Proc. 12th Int. Conf. on Very Large Data Bases*, pages 149–159, 1986.

[63] Sergio Marti, Prasanna Ganesan, and Hector Garcia-Molina. DHT routing using social links. In *Peer-to-Peer Systems, First International Workshop*, pages 100–111, 2004.

[64] P. Maymounkov and D. Maziéres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. First International Workshop on Peer-to-Peer Systems, IPTPS 2002*, pages 53–65, 2002.

[65] S. Michel, P. Triantafillou, and G. Weikum. KLEE: A framework for distributed top-k query algorithms. Technical report, Max-Planck Institute for Computer Science, Germany, and University of Patras, Greece, 2005.

[66] S. Michel, P. Triantafillou, and G. Weikum. Minervainfinity: A scalable efficient peer-to-peer search engine. In *Middleware 2005, IFIP/ACM International Conference on Distributed Systems Platforms*, pages 60–81, 2005.

[67] David L. Mills. *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, 2006.

[68] A. Mondal, Y. Lifu, and M. Kitsuregawa. P2PR-Tree: An R-tree-based spatial index for peer-to-peer environments. In *EDBT Workshops*, pages 516–525, 2004.

[69] Priya Narasimhan, Tudor Dumitras, Aaron M. Paulos, Soila M. Pertet, Carlos F. Reverte, Joseph G. Slember, and Deepti Srivastava. Mead: support for real-time fault-tolerant CORBA. *Concurrency - Practice and Experience*, 17(12):1527–1545, 2005.

[70] S. Nath, P. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (Sensys)*, pages 250–262, 2004.

[71] W. Nejdl, W. Siberski, U. Thaden, and W. Balke. Top-k query evaluation for schema-based peer-to-peer networks. In *International Semantic Web Conference*, pages 137–151, 2004.

[72] Frank Olken and Doron Rotem. Random sampling from B+ trees. In *Proc. 15th Int. Conf. on Very Large Data Bases*, pages 269–277, 1989.

[73] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly Media, 2001.

[74] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Second Edition*. Prentice-Hall, 1999.

[75] V. Papadimos, D. Maier, and K. Tufte. Distributed query processing and catalogs for peer-to-peer systems. In *First Biennial Conference on Innovative Data Systems Research*, 2003.

[76] M. Perpinan. A review of dimension reduction techniques. Technical Report CS-96-09, University of Sheffield, 1997.

[77] Theoni Pitoura and Peter Triantafillou. Self-join size estimation in P2P data systems. In *Proc. Int. Conf. on Data Engineering*, 2008.

[78] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Algorithms and Data Structures, Workshop WADS '89*, pages 437–449, 1989.

[79] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief announcement: prefix hash tree. In *Proc. ACM SIGACT-SIGOPS Symp. on Principles of Dist. Comp.*, 2004.

[80] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2002.

[81] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. In *Proc. ACM SIGCOMM*, pages 331–342.

[82] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, pages 161–172, 2001.

[83] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. 18th IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, 2001.

[84] Cristina Schmidt and Manish Parashar. Enabling flexible queries with guarantees in P2P systems. *IEEE Internet Computing*, 8(3):19–26, 2004.

[85] D. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization*. Wiley-Sons, 1992.

[86] William Stallings. *Operating Systems: Internals and Design Principles*. Prentice Hall, 2004.

[87] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. In *Proc. ACM SIGCOMM*, pages 149–160, 2001.

[88] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proc. ACM SIGCOMM*, pages 175–186, 2003.

[89] the National Institute of Standards and Technology. Secure hash standard. http://www.itl.nist.gov/fipspubs/fip180-1.htm, August 2002.

[90] P. Triantafillou and T. Pitoura. Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In *Proc. First International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, pages 169–183, 2003.

[91] R. v. Renesse and A. Bozdog. Willow: DHT, aggregation, and publish/subscribe in one protocol. In *Peer-to-Peer Systems III, Third International Workshop (IPTPS)*, pages 173–183, 2004.

[92] P. Valduriez and G. Gardarin. Join and semijoin algorithms for a multiprocessor database machine. *ACM Trans. Database Syst.*, 9(1):133–161, 1984.

[93] P. Valduriez and E. Pacitti. Data management in large-scale P2P systems. In *High Performance Computing for Computational Science - VECPAR 2004, 6th International Conference*, pages 104–118, 2004.

[94] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2), May 2003.

[95] Q. Wang, A. K. Jha, and M. T. Özsu. An XML routing synopsis for unstructured P2P networks. In *the First International Workshop on XML, Web, and Internet Contents Technologies*, pages 176–183, 2006.

[96] Qiang Wang and M. Tamer Özsu. An efficient eigenvalue-based P2P XML routing framework. In *Seventh IEEE International Conference on Peer-to-Peer computing*, pages 105–112, 2007.

[97] G. Weikum. The Web in ten years: Challenges and opportunities for database research. In *Decimo Convegno Nazionale su Sistemi Evoluti per Basi di Dati*, pages 3–15, 2002.

[98] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *Proc. ACM SIGCOMM*, pages 379–390, 2004.

[99] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proc. 22nd Int. Conf. on Distributed Computing Systems*, pages 5–12, 2002.

[100] M. Zaharia and S. Keshav. Adaptive peer-to-peer search. Technical Report 2004-55, University of Waterloo, November 2004.

[101] C. Zhang, A. Krishnamurthy, and R. Y. Wang. Brushwood: distributed trees in peer-to-peer systems. In *IPTPS*, pages 47–57, 2005.

[102] R. Zhang and Y. C. Hu. Assisted peer-to-peer search with partial indexing. In *The 24st Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1514–1525, 2005.

[103] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of California, Berkeley, 2001.